НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

До захисту допущено:

В.о. завідувача кафедри

_____ Оксана ТИМОЩУК

«___»_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системи та методи штучного інтелекту»

спеціальності 122 «Комп'ютерні науки та інформаційні технології»

на тему: «Система покращення якості зображення з використанням генеративної змагальної мережі»

Виконала:
студентка IV курсу, групи КА-65
Єремєєва Вероніка Геннадіївна _____

Керівник:
професор
Зайченко Олена Юріївна _____

Консультант з економічного розділу:
доцент, д.е.н.
Шевчук Олена Анатоліївна _____

Консультант з нормоконтролю:
доцент, к.т.н.
Коваленко Анатолій Єпіфанович _____

Рецензент:
к.т.н.,с.н.с. кафедри ПЗКС ФПМ
Вішталь Дмитро Іванович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 122 «Комп'ютерні науки та інформаційні технології»
Освітньо-професійна програма «Системи та методи штучного інтелекту»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Оксана ТИМОЩУК
«25» травня 2020 р.

ЗАВДАННЯ

на дипломну роботу студенту
Єремєєвій Вероніці Геннадіївні

1. Тема роботи «Система покращення якості зображення з використанням генеративної змагальної мережі», керівник роботи Зайченко Олена Юріївна, професор, затверджені наказом по університету від «25» травня 2020 р. № 1143-с

2. Термін подання студентом роботи 8.06.2020.

3. Вихідні дані до роботи масштабоване зображення високої роздільної здатності.

4. Зміст роботи аналіз існуючих методів масштабування зображень, ознайомлення зі структурою згорткових нейронних мереж та визначення архітектури для генеративної змагальної мережі, проведення порівняльного аналізу проміжних результатів системи.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо) зображення вхідних та вихідних даних, архітектур моделей, графіків функцій.

6. Консультанти розділів роботи*

Розділ		Підпис, дата

_____

	Прізвище, ініціали та посада консультанта	завдання видав	завдання прийняв
Економічний	Шевчук О.А., доцент		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Формулювання теми БДР	03.09.2019	
2.	Огляд літератури з теми роботи та її опрацювання	03.09.2019 – 01.03.2020	
3.	Програмна реалізація системи	01.03.2020 – 20.04.2020	
4.	Ознайомлення з ДСТУ 3008-96 та стандартами ЄСПД	20.04.2020 - 27.04.2020	
5.	Написання першого розділу БДР	28.04.2020 – 03.05.2020	
6.	Написання другого розділу БДР	03.05.2020 – 07.05.2020	
7.	Написання третього розділу БДР	14.05.2020 – 27.05.2020	
8.	Написання четвертого розділу БДР та оформлення роботи	28.05.2020 – 07.06.2020	

Студент Вероніка ЄРЕМЄЄВА

Керівник Олена ЗАЙЧЕНКО

РЕФЕРАТ

Дана дипломна робота містить 85 ст., 4ч., 6 табл., 39 рис., 2 дод., 16 джерел.

МАСШТАБУВАННЯ ЗОБРАЖЕНЬ, ГЕНЕРАТИВНО-ЗМАГАЛЬНА МЕРЕЖА, ЗГОРТКОВА МЕРЕЖА, МЕТОДИ ІНТЕРПОЛЯЦІЇ ЗОБРАЖЕНЬ, НЕЙРОННІ МЕРЕЖІ, МЕТОД ЗВОРОТНЬОГО ПОШИРЕННЯ ПОМИЛКИ, ПЕРЦЕПТИВНА ФУНКЦІЯ ВИТРАТ

Об’єкт дослідження – зображення низької роздільної здатності або малого розміру.

Мета роботи – дослідити існуючі методи покращення якості зображень, створити власну модель SRGAN.

Метод дослідження – аналіз існуючих методів інтерполяції зображення, створення власної моделі та аналіз результатів генеративної змагальної мережі.

Результатом роботи є модель генеративної змагальної мережі, а саме система, що покращує роздільну здатність вхідного зображення.

Новизною роботи є створення даної системи у кросплатформній оболонці, що дозволяє використовувати потужності віртуального комп’ютера.

Галузь застосування: проведена робота може бути використана при підвищенні роздільної здатності зображень, особливо в таких областях, як відеоспостереження, відновлення старих фотокарток та медична діагностика.

Програмний продукт реалізований за допомогою мови програмування Python. Під час дослідження було проведено аналіз проміжних результатів системи та порівняння вихідних зображень.

ABSTRACT

This thesis contains 85 p., 4 ch., 6 tab., 39 fig., 2 app., 16 src..

IMAGE SCALING, GENERATIVE ADVESARIAL NETWORK, CONVOLUTIONAL NETWORK, IMAGE INTERPOLATION METHODS, NEUREAL NETWORKS, BACKPROPAGATION, PERSEPTUAL LOSS

Object of research: low resolution image or small image.

The purpose of the work is to explore existing methods of improving image quality and to create our own SRGAN model.

Research method: analyzing existing methods of image interpolation and creation of generative adversarial network model and checking the results of the model.

The result of the work is a SRGAN model which improves the quality of the image.

Novelty of this study is creating model in crossplatform system which allows using power of a virtual computer.

Using scope: the work can be used to increase the resolution on an image. Especially, in areas such as video monitoring, restoration old media files and medical diagnostics.

This product was made using Python. During the study, analysis of intermediate results were made as well as comparison of the output images.

ЗМІСТ

ВСТУП

Діджиталізація та використання все більшого об’єму даних призводить до питання якості цифрових файлів. Будь то відео, фото або намальована картинка. Чим більша кількість пристроїв, тим різноманітніші екрани приладів. Та задля того, щоб на кожному з них зображення відображалось гарно, деколи використовуються алгоритми масштабування зображень.

В наш час технології розпізнавання образів досягли великих успіхів, особливо у сфері глибокого навчання. Побудова складних ієрархічних моделей дозволяє вирішувати різноманітні задачі. Великого прориву досягли дискримінативні моделі, що визначають приналежність об’єкта до заздалегідь відомих класів даних. С таким підходом можна провести аналогію: людина навчається відрізняти предмети один від одного, чим пізнає світ.

Існують також альтернативна концепція машинного навчання, що базується на використанні генеративних моделей (generative). На відміну від дискримінативної моделі, базовий принцип роботи такої моделі можна сформулювати як «зрозуміти – означає повторити». Тобто якщо система добре розуміє предмет, то йому не буде важно створити копію даного предмету.

У даній роботі буде розглядатися гібрид – Generative Adversarial Network (GAN), що об’єднує дискримінативну та генеративну моделі. Сутністю даного алгоритму є гра, тобто змагання генеративної та дискримінативної моделі, в ході якої кожна з них буде навчатись за рахунок свого супротивника.

Отже, дана робота присвячена аналізу та впровадженню генеративної змагальної мережі для покращення роздільної здатності зображення. Другим етапом є дослідження програмної структури нейронної системи та аналіз отриманих результатів.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність задачі

На сьогоднішній день темпи розвитку програмних продуктів набагато випереджають розвиток фізичних технологій, апаратних комплексів та систем. Підвищення якості зображення, тобто його роздільної здатності може бути реалізована завдяки техніці зі спеціальними лінзами. Але треба зауважити, що вона потребує великих капіталовкладень.

Під підвищенням якості зображення мається на увазі підвищення його інформативності, тобто збільшення масштабу зображення без великих втрат візуальної інформації.

Наразі існує купа пристроїв із різними екранами та підлаштовуватись під потреби кожного користувача дуже складно. Виробники відео- та фото- матеріалів не можуть створювати свій продукт під кожен екран. Тому з’явилася така потреба у відповідних алгоритмах та програмах, які будуть масштабувати зображення відповідно до вихідного пристрою. Так, виробники зможуть задовольнити потреби користувачів. Проблема масштабування зображення або інтерполяції, стала необхідною та поступово почали створюватись відповідні алгоритми з використанням нейронних мереж, які дають гарні результати.

Використання інтерполяції зображення може бути в абсолютно різних сферах: у системах безпеки задля збільшення зображення та отримання більш чітких ознак машин, людей та інших деталей; кінематограф та виробництво мультфільмів може використовувати масштабування зображення для відновлення старих плівок без великих втрат якості; не є виключенням і повсякденне життя, де люди зможуть відновлювати свої старі зображення щоб зберегти згадки зі своєї історії. Навіть космічні станції використовують алгоритми масштабування, щоб зробити якомога більше наближення та отримати знімок максимально можливої чіткості з космосу.

1.2 Аналіз існуючих алгоритмів

Найпростішим методом підвищення роздільної здатності зображення є інтерполяція. При інтерполюванні зображення має вигляд функції, де пікселі зображення – це точки, в яких відомі значення функції. Процес інтерполяції – це знаходження нових значень по заданому дискретному набору даних. Точне відновлення інформації шляхом інтерполяції неможливо.

При вирішенні задач збільшення роздільної здатності будується перехід з більш глибокої сітки до мілкої сітки, що ще називають ресемплінгом. А коефіцієнтом масштабування зображення є відношення шагу крупної сітки до шагу малої. Найпоширенішими методами інтерполяції зображення є метод найближчого сусіда, білінійна, бікубічна інтерполяція та фільтр Ланцоша. Будь-який метод інтерполяції балансує між трьома ефектами: розмиття, аліасинг (ефект «ступінчастості») та освітленості країв.

Так як звичайні методи інтерполяції не можуть розв’язати задачу так, щоб не втратити показник якості одного з ефектів, для задач масштабування зображень використовуються нейронні мережі з супер розширенням зображень, де на вхід подається зображення з низькою роздільною здатністю, масштабується в мережі та обчислюються значення пікселів, які треба доповнити. У цій роботі буде йтись мова про алгоритм SRGAN, але є ще декілька інших методів.

SRCNN: це конволюційна нейромережа, що складається з трьох шарів згортки: вилучення та представлення патчів, нелінійне відображення та реконструкція.

VDSR: Глибока мережа, що використовує аналогічну структуру як і SRCNN, але для досягнення більшої точності йде глибше.

Отже, найпростіші методи інтерполяції зображення можуть застосовуватись для підвищення роздільної здатності, але треба зрозуміти, що вони не вносять нової

інформації до зображення. А при супер розширенні використовується інформація одразу з декількох зображень, що дозволяє вносити в результуюче зображення більш точні результати.

1.3 Особливості предметної області

Як було вже зазначено, інтерполяція зазвичай балансує між трьома недоліками: аліасинг, розмиття та ефекту Гіббіса.

Аліасинг - ступінчастість або нерівність. Приклад аліасингу на рис. 1.1 та 1.2.

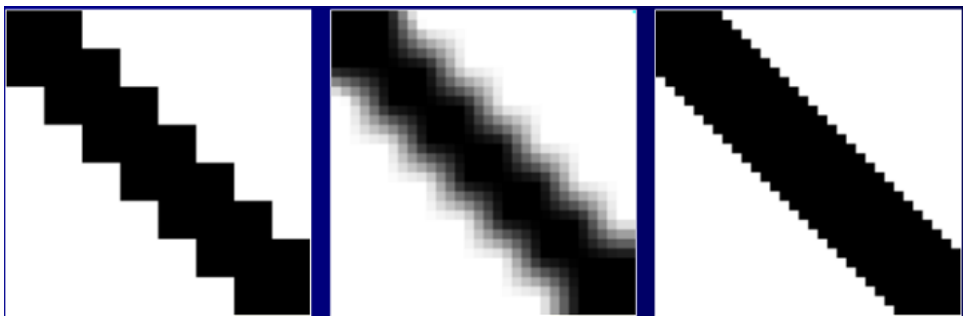


Рисунок 1.1 – Приклад дефекту аліасингу

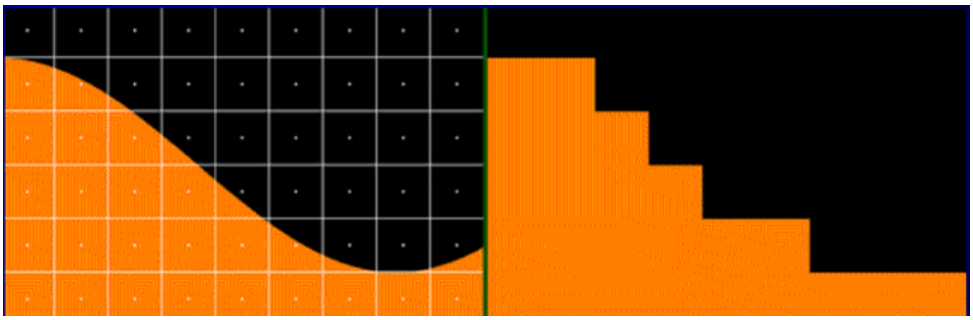


Рисунок 1.2 – Приклад аліасингу

Ефект Гіббса – негативний ефект, що виникає при інтерполяції. На зображенні проявляється у вигляді різкого шуму біля різких перепадів інтенсивності зображення. Приклад ефекту Гіббса на рис. 1.3.

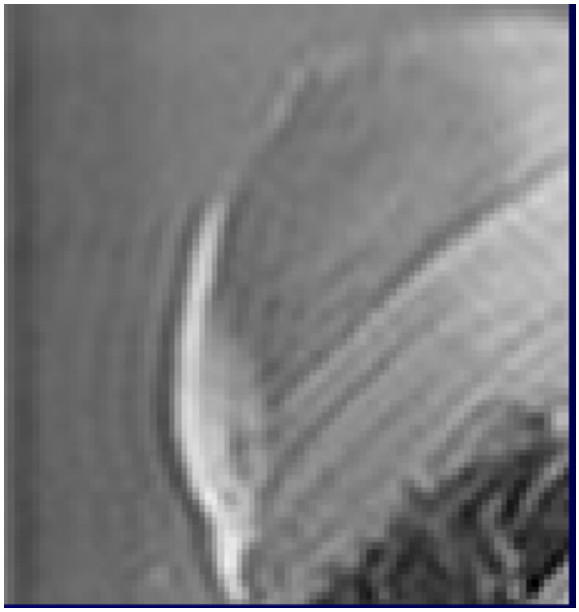


Рисунок 1.3 – Приклад ефекту Гіббса

Тож у даній роботі ми будемо мінімізувати ці дефекти та досліджувати як буде змінюватись якість зображення з різною ступеню натренованості мережі.

1.4 Висновки до першого розділу

У даному розділі було розглянуто загальні методи масштабування та їхні недоліки, які треба мінімізувати при створенні власної системи. Було розглянуто два основних типи вирішення проблем масштабування зображення: за допомогою інтерполяції та супер розширенням (супер роздільності). Тож методи інтерполяції працюють не погано, але вони не можуть відтворити зображення з великою точністю. На противагу цьому, вони є дуже швидкими алгоритмами. Супер розширення може аналізувати зображення краще, тому всі три існуючі недоліки мінімізуються у цих алгоритмах.

2 МЕТОДИ ГЛИБИННОГО НАВЧАННЯ ДЛЯ ПОКРАЩЕННЯ ЯКОСТІ ЗОБРАЖЕННЯ

2.1 Основні методи інтерполяції зображень

2.1.1 Метод найближчого сусіда

Метод вважається базовим зі всіх алгоритмів інтерполяції. Цей метод потребує найменшої кількості часу, бо в алгоритмі враховується тільки один піксель, що є ближнім до точки інтерполяції.

Використовуючи метод найближчого сусіда зображення буде низької якості, бо результатом даного методу є зображення, в якому кожний піксель стає більшим. Це можна побачити на прикладі рис.2.1.

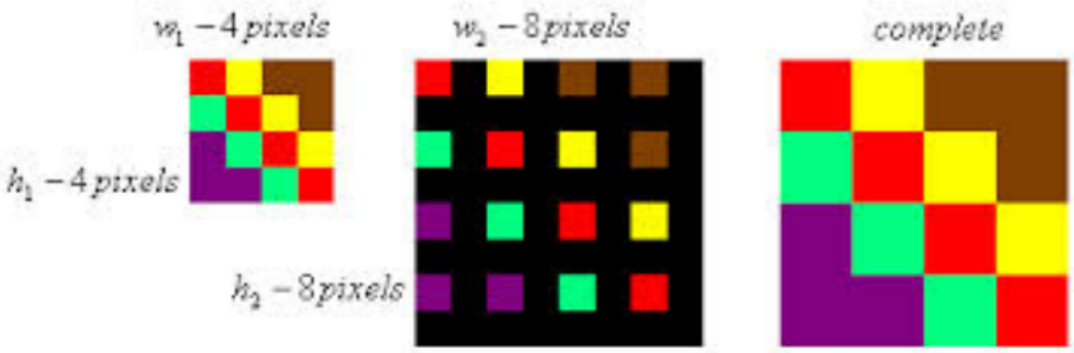


Рисунок 2.1 – Приклад роботи алгоритму найближчого сусіда

Отже, даний алгоритм є швидким, але зазнає великих страт у якості зображення.

2.1.2 Білінійна інтерполяція

Білінійна інтерполяція використовує більше інформації, тому й зображення на виході є більш гладким, порівняно з алгоритмом найближчого сусіда. Але збільшується час обробки інформації, так як складність алгоритму більша.

Білінійна інтерполяція аналізує квадрат 2x2 відомих пікселів, що розташовані навколо невідомого. В якості значення, що інтерполюється, використовується зважене усереднене значення цих чотирьох пікселів.

Як працює білінійна інтерполяція розглянемо на рис. 2.2 та рис. 2.3.

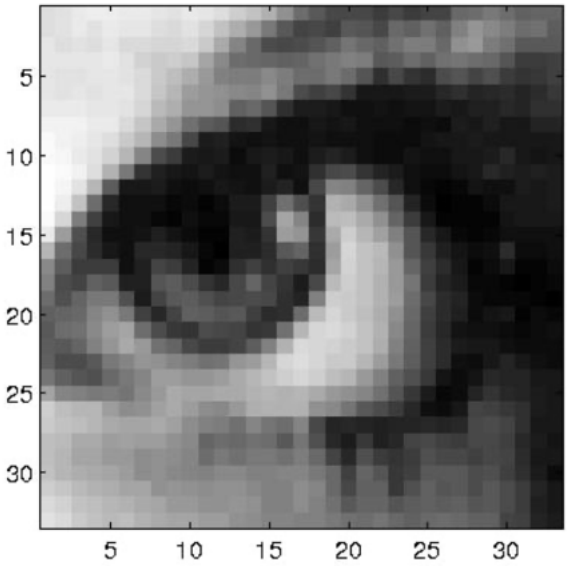


Рисунок 2.2 – Вхідні дані для білінійної інтерполяції

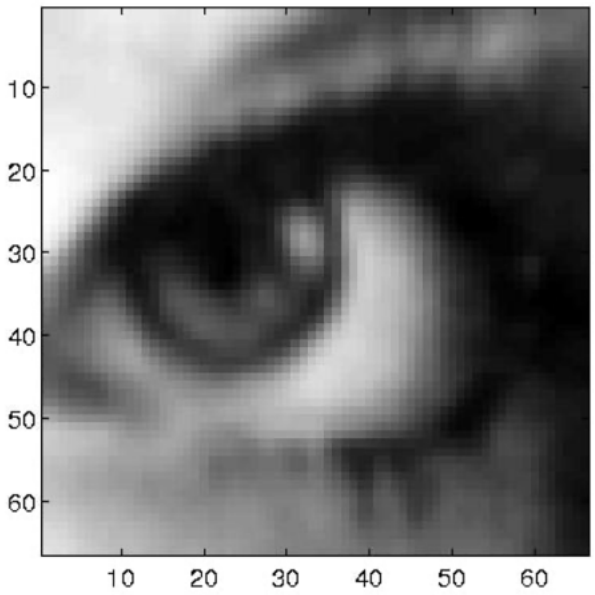


Рисунок 2.3 – Результат білінійної інтерполяції

2.1.3 Бікубічна інтерполяція

Бікубічна інтерполяція дещо схожа на білінійну, але вона бере більший масив даних до уваги. Так, метод інтерполює масив 4x4 пікселів, що знаходяться навколо невідомого. Важливо зауважити, що пікселі знаходяться на різній відстані від невідомого, тому мають різну вагу: найближчі – більшу, крайні – меншу.

На відміну від попередніх методів, даний отримує більш різке зображення на виході. Можливо тому цей алгоритм став популярним та використовується у деяких програмах обробки зображень.

Приклад роботи бікубічної інтерполяції можемо побачити на рис. 2.4 та 2.5.

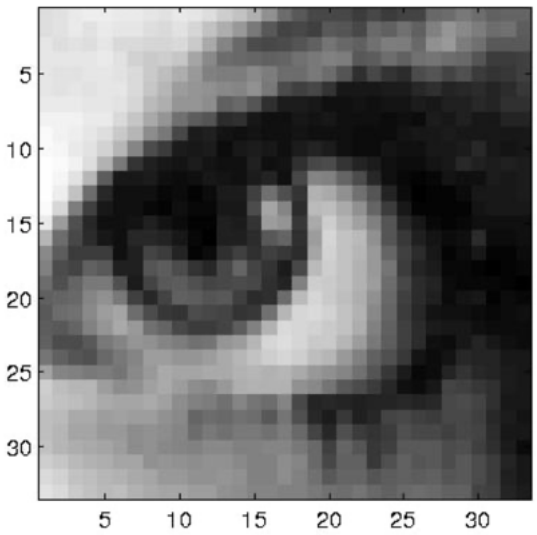


Рисунок 2.4 – Початкове зображення

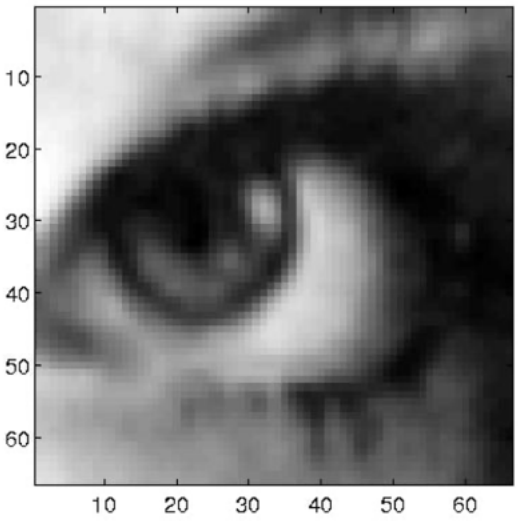


Рисунок 2.5 – Отримане зображення бікубічним інтерполюванням

2.1.4 Фільтр Ланцоша

Фільтр Ланцоша використовується для інтерполяції значення цифрового сигналу між його зразками. Він відображає кожен зразок даного сигналу до зведеної та масштабованої копії ядра Ланцоша. Ядро Ланцоша є віконною sinc-функцією. Фільтр Ланцоша використовується для збільшення частоти дискретизації цифрового сигналу.

Sinc-функція обчислюється за формулою 2.1:

$$sinc(x) = \begin{cases} \frac{\sin(\pi x)}{\pi x}, x \neq 0 \\ 1; x = 0 \end{cases} \quad (2.1)$$

Фільтр Ланцоша отримується помножуючи вікно Ланцоша на sinc-функцію.

Алгоритм Ланцоша дещо схожий до кубічної інтерполяції. Але відмінність метода в тому, шо він може обробляти детальну графіку без ефекту розмиття. На противагу цього, даний метод має більшу кількість обчислень, бо використовується 36 пікселів для інтерполяції вхідного значення.

2.2 Нейронні мережі

Штучні нейронні мережі є обчислювальними алгоритмами. Такий алгоритм створений, щоб імітувати поведінку біологічних систем, що складаються з нейронів. Нейрони у мозку людини обробляють сигнали та відправляє їх у вигляді електричних сигналів. Нейрони зв’язані між собою спеціальною структурою – синапсами, які дозволяють передавати сигнали між собою. Та завдяки нейронам та їхнім зв’язкам, формується нейронна мережа.

Штучна нейронна мережа – метод обробки інформації, який створений подібно до людської нейронної мережу. Але замість передавання та моделювання електричних сигналів, будуть використовуватись числа. Вони будуть подаватись на вхід мережі та проходячи по мережі, будуть відповідним чином змінюватись. А на виході ми отримаємо одне результуюче число.

Кожен зв’язок нейронів буде характеризуватись відповідним числом, яке називається вагами. Сигнал, що буде проходити через зв’язок буде помножений на вагу даного зв’язку. На рис. 2.6 можем побачити, що у зв’язків (чорних стрілок) є відповідні ваги. На рисунку не зображені всі, але вони взагалі є у кожного зв’язку.

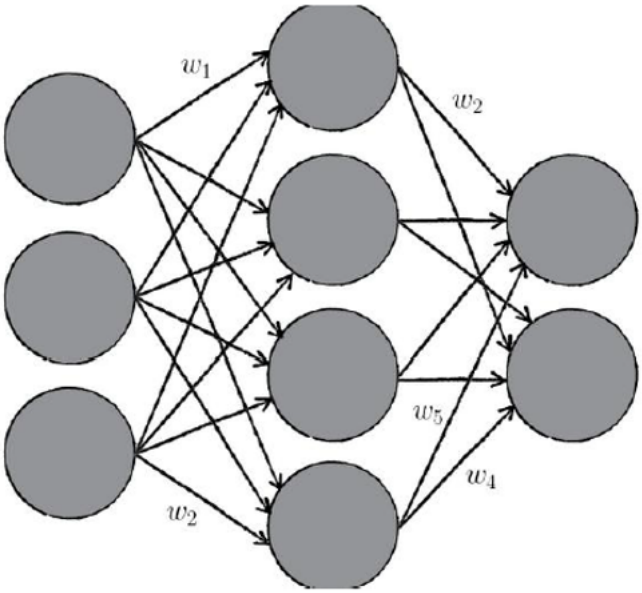


Рисунок 2.6 - Зв’язок штучних нейронів

Штучна нейронна мережа зазвичай має декілька шарів. Кожен шар містить множину взаємопов’язаних вузлів, які містять функцію активації. Нейронна мережа може мати такі типи шарів: вхідний, прихований, вихідний шари.

Структуру штучного нейрона можна побачити на рис 2.7

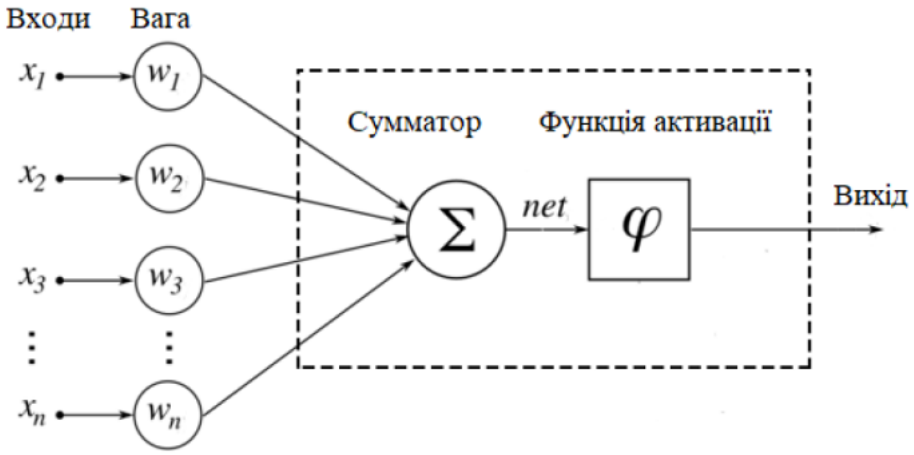


Рисунок 2.7 – Модель штучного нейрона

Кожен нейрон має входи, через які він приймає сигнал. Коли сигнал входить, то одразу множиться на свою вагу. Далі всі ці добутки вхідних сигналів на вагу $x_i * w_i$ передаються у суматор, який сумує всі отримані добутки. Результатом роботи суматора є число, яке називають зваженою сумою.

Подавати зважену суму на вихід немає сенсу. Нейрон повинен обробити сю суму та сформувати вихідний сигнал. Для цього використовують функцію активації. Вона перетворює зважену суму у число, що є виходом нейрона. Для різних типів нейронів використовують різні функції активації.

2.2.1 Ступінчаста функція

Функція єдиного скачка вважається найпростішим видом функції активації. Вихідне значення нейрона може бути або 0, або 1. Якщо зважена сума більше якогось зазначеного параметру, то вихід нейрона буде дорівнювати 1, якщо нижче, то 0. Графік даної функції зображено на рис.2.8.

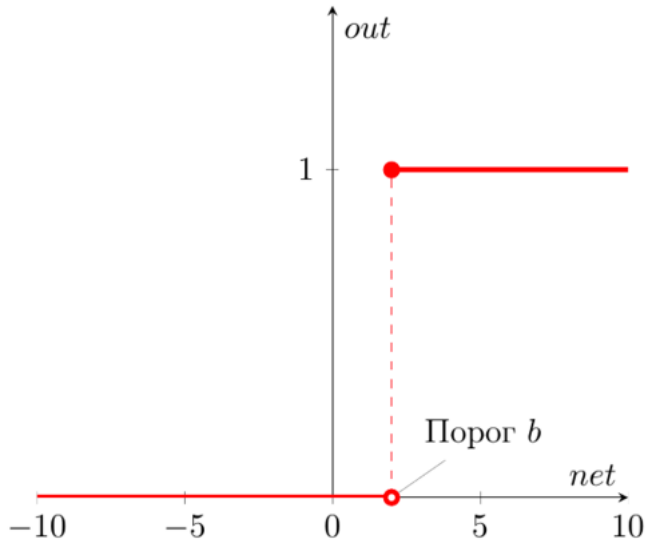


Рисунок 2.8 - Графік ступінчастої функції з порогом b

Математично записується як:

$$out(net) = \begin{cases} 0, net < b \\ 1, net \geq b \end{cases} \quad (2.2)$$

Дана функція активація буде гарно працювати, якщо у нас буде лише один клас для класифікації та два варіанта – 0 або 1. Але якщо треба класифікувати декілька класів, тоді при активації декількох нейронів ми ніяк не зрозуміємо який саме клас нам підходить найбільше до об’єкту.

Якщо функція активації не буде бінарною, тоді ми зможемо отримувати різні значення функції та обирати той нейрон, у якого більше значення активації. Так можна буде класифікувати декілька класів об’єктів та отримувати максимально точний результат.

2.2.2 Лінійна функція активації

Лінійна функція – пряма лінія, що пропорційна входу (зваженій сумі на даному нейроні). Така функція вже не є бінарною, тому можна отримати спектр значень функції активації. Графік даної функції зображено на рис.2.9.

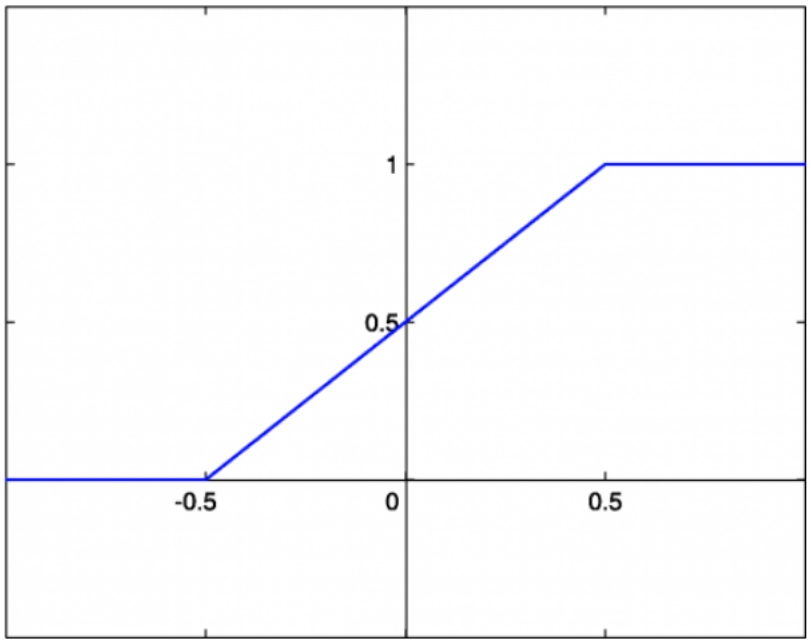


Рисунок 2.9 - Графік лінійної функції активації

Математично записується як:

$$out(net) = c * x \tag{2.3}$$

Так як похідна цієї функції – константа, то градієнт є постійним вектором, а спуск виконується по постійному градієнту. Якщо система робить хибне передбачення, то зміни, зроблені методом зворотнім розповсюдженням помилки, будуть також постійними та не будуть залежати від змін на вході.

Досить великим недоліком лінійної функції активації є те, що при використанні їх на декількох шарах нейронної системи не буде мати значення. На виході ми все одно отримаємо лінійну функцію активації від виходів на першому шарі. Тому тут немає сенсу робити декілька шарів, бо вони можуть бути замінені одним шаром.

Лінійна функція активації використовується досить рідко у задачах класифікації.

2.2.3 Сигмоїдальна функція

Сигмоїдальна функція виглядає гладкою та дещо подібна до ступінчастої функції. . Графік даної функції зображено на рис.2.10.

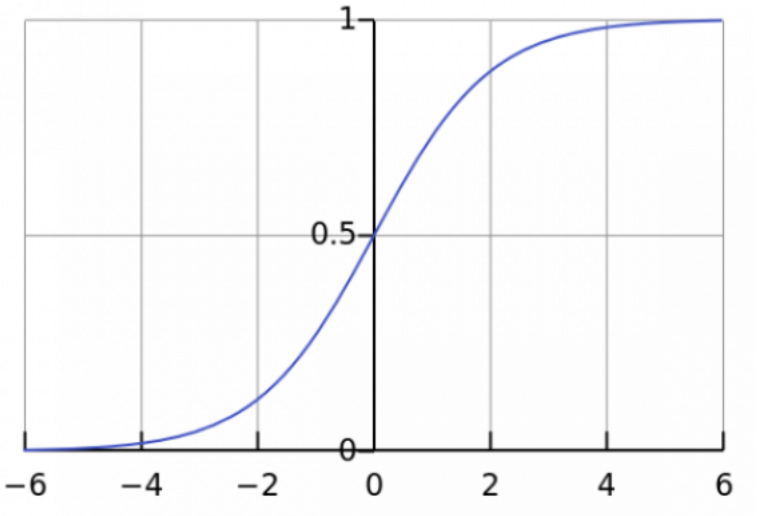


Рисунок 2.10 - Графік сигмоїдальної функції

Математично записується таким чином:

$$A = \frac{1}{1 + e^{-x}} \tag{2.4}$$

Сигмоїдальна функція є нелінійною, тому комбінація таких функцій призводить також до нелінійної функції. Тому можна використовувати її у багатошаровій нейронній мережі. Функція не є бінарною та для неї характерний гладкий градієнт. В діапазоні $x \in [-2; 2]$ значення Y змінюється дуже швидко, що вказує на те, що будь-яка невелика зміна значення X в цій області несе за собою велику зміну значення Y.

Наразі сигмоїдальна функція є однією з найбільш часто застосованих функцій активації в нейромережах.

Недоліком є те, що при наближенні до кінців сигмоїди значення Y буде слабо реагувати на зміну X. Це приведе до того, що градієнт буде мати маленьке значення в цих областях, що в свою чергу може призвести до проблеми з градієнтом зникнення. Тоді нейронна мережа перестає навчатись, або робить це дуже повільно.

2.2.4 Гіперболічний тангенс

Гіперболічний тангенс є скорегованою сигмоїдальною функцією. . Графік даної функції зображено на рис.2.11.

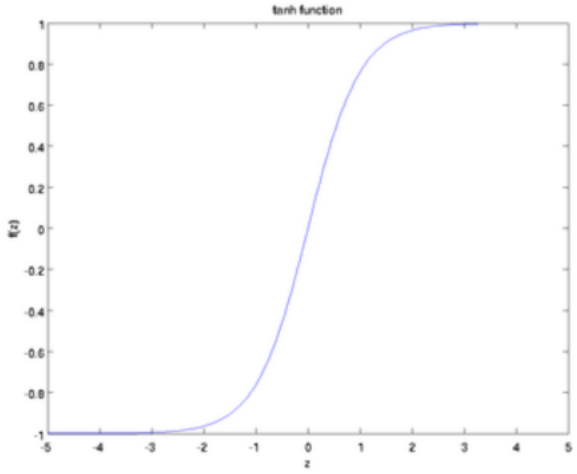


Рисунок 2.11 - Графік функції гіперболічного тангенсу

Математично описується таким чином:

$$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \quad (2.5)$$

Функція має такі ж характеристики як і сигмоїдальна функція. Але градієнт у тангенціальної функції більший. Так як і сигмоїдальній функції, гіперболічному тангенсу присутня проблема зникнення градієнту.

2.2.5 ReLu

ReLu (rectified linear unit) повертає значення x, якщо x позитивне, та 0 в інших випадках. Графік даної функції зображено на рис.2.12.

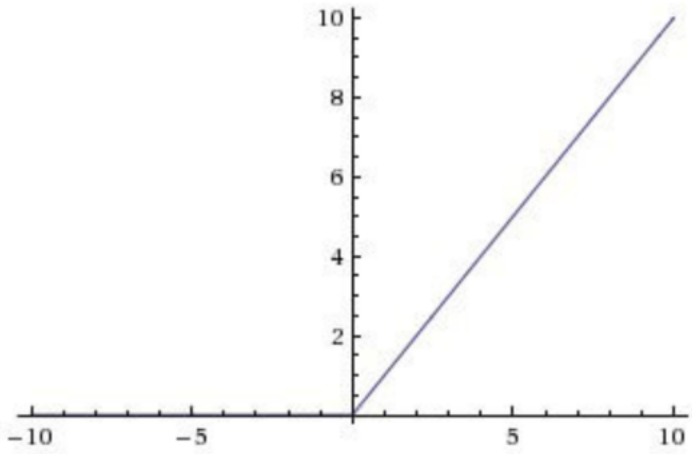


Рисунок 2.12 - Графік ReLu

Математично має вигляд:

$$A(x) = \max(0, x) \quad (2.6)$$

На перший погляд може здатися, що дана функція має схожі проблеми, що й лінійна функція активації. Але ReLu нелінійна та комбінація даних функцій також буде нелінійною.

Використання сигмоїди або гіперболічного тангенсу призводить до активації всіх нейронів аналоговим способом. Мається на увазі, що майже всі активації будуть оброблені для опису виходу мережі. Це є досить затратно. В ідеалі було б зробити так, щоб деякі нейрони були не активовані, тоді б активація була більш ефективною. Але завдяки ReLu можна так зробити. Тому що частина активацій буде рівна 0 через від'ємне значення x.

Так як ReLu – горизонтальна лінія при від'ємних значеннях x, то градієнт цієї частини буде дорівнювати нулю. Через це, ваги не будуть корегуватись під час спуску. Тобто нейрони не будуть реагувати на зміни в помилці, бо градієнт рівний нулю. Це називають проблемою вмираючого ReLu. Через це деякі нейрони не будуть відповідати та просто «вимкнуться».

2.2.6 Етапи навчання нейронних мереж

Навчання в нейронних мережах відбувається в два етапи:

- Пряме розповсюдження помилки
- Зворотне розповсюдження помилки

Під час прямого розповсюдження помилки робиться передбачення результату. При зворотному розповсюдженні похибка між фактичною відповіддю та передбаченим результатом мінімізується. Алгоритм зображено на рис.2.13 та 2.14.

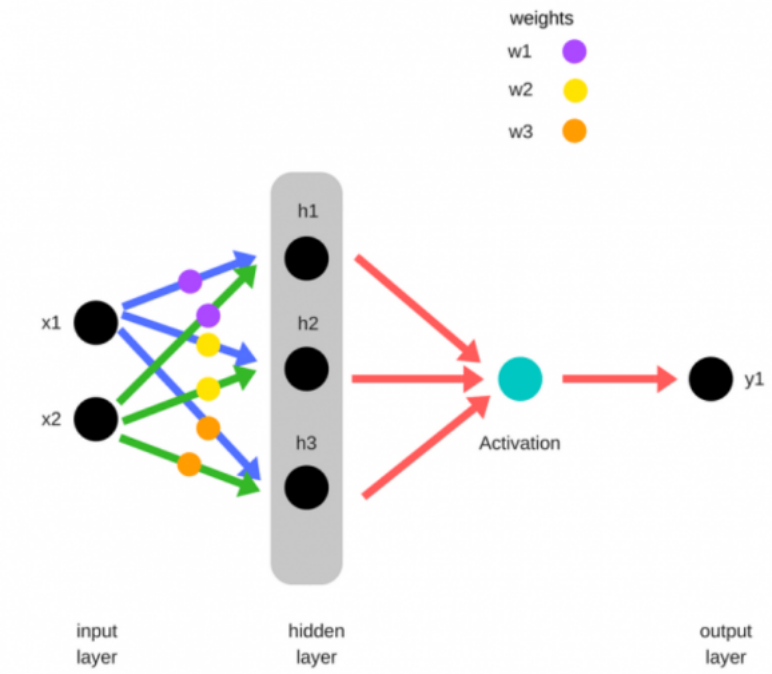


Рисунок 2.13 - Візуалізація прямого розповсюдження помилки

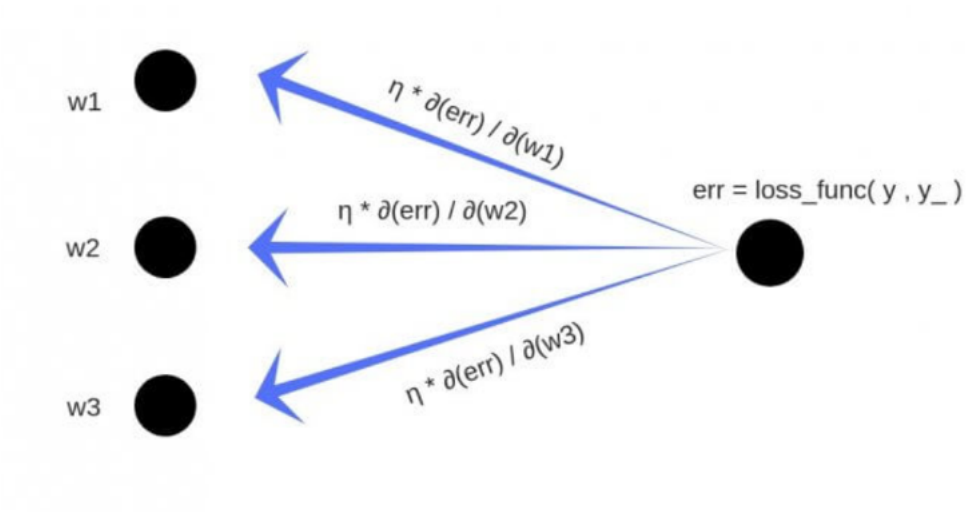


Рисунок 2.14 - Візуалізація зворотного розповсюдження помилки

Сумарна помилка обчислюється як різниця між фактичним значенням з набору даних та отриманим системою результатом. Часткова похідна помилки обчислюється по кожній вазі та ці диференціали вказують на вклад кожної ваги на загальну помилку. Далі ці диференціали множаться на число, яке називають швидкістю навчання або learning rate.

2.3 Згорткові нейронні мережі

Звичайні нейронні мережі зазвичай не використовуються для роботи в області комп'ютерного зору, тому що з'являється дуже велика складність обчислення. Так як при використанні глибоких нейронних мереж модель потребувала б більшої кількості ваг та дуже великих добутків матриць, що потребує великих обчислювальних потужностей. Коли була представлена згорткова нейронна мережа, то було виявлено, що вона більш ефективніша та швидша в задачах комп'ютерного зору.

Згорткові нейронні мережі або CNN (Convolutional Neural Network) – вид нейронної мережі, де хоч на одному слою буде використана операція згортки. Наразі ця нейронна мережа дуже часто використовується для задач розпізнавання облич та аналізу текстів.

Згорткові нейронні мережі забезпечують часткову стійкість до зміни масштабу, зсувів, поворотів та зміни ракурсу. Тому є одним з найбільш популярних методів.

2.3.1 Операція згортки та вхідні дані

Згортка може бути двох типів: лінійна та дискретна. Але так як дані в комп'ютерах дискретні та заміри виконуються з деяким інтервалом, тому зазвичай розглядають дискретну згортку.

Згортка може бути описана формулою

$$(f \times g)[m,n] = \sum_{k,l} f[m-k, n-l] \cdot g[k,l], \tag{2.7}$$

де f – вихідна матриця зображення,

g – ядро (матриця згортки).

Так як вхідним даним є зображення, то вони представляються на вході у вигляді матриці з глибиною – 3, тобто розбивається на три канали: синій, червоний, зелений. Матриця також може бути одна, якщо зображення в відтінках сірого.

Кожен піксель вхідного зображення потрібно нормалізувати по формулі:

$$f(p, min, \max) = \frac{p-min}{max-min},$$

де f – функція нормалізації,

p – значення кольору від 0 до 255,

min – мінімальне значення пікселя – 0,

max – максимальне значення пікселя.

Згортка – математична операція, що приймає два параметри входу:

1. Матриця зображення з параметрами об’єму (висота * ширина * глибина).

2. Фільтр з параметрами висоти, ширини та глибини.

На рис. 2.15 можемо подивитись результат з отриманих даних.

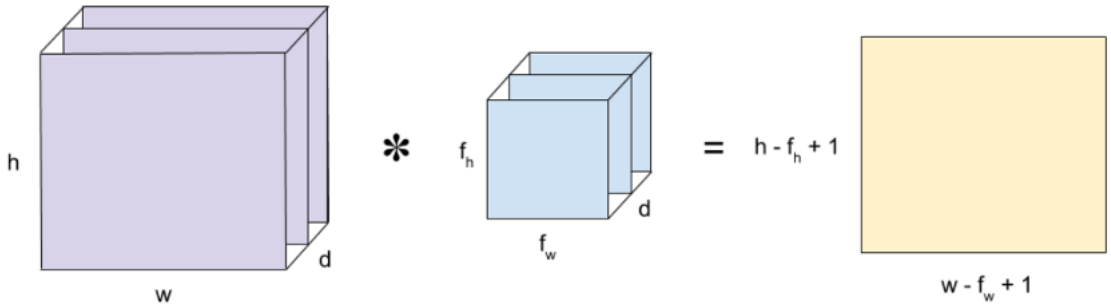


Рисунок 2.15 – Згортка зображення з глибиною 3

Результат згортки, наприклад, для верхнього лівого пікселя рахується наступним чином: спочатку фільтр розміщується у лівий кут вхідного зображення, далі підраховується поелементний добуток кожного пікселя та підсумовуються ці добутки, щоб отримати значення для конкретного пікселя виходу.

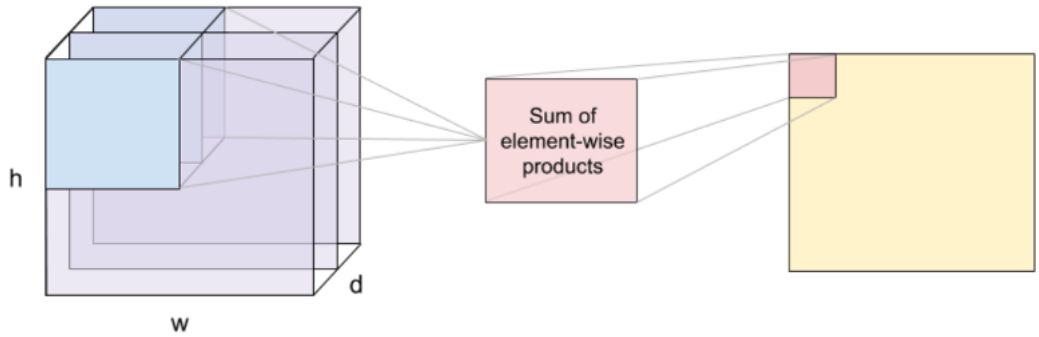


Рисунок 2.16 – Згортка зображення попіксельно

Для проведення згортки значення глибини зображення та глибини фільтра повинні співпадати.

Операція, яку описано вище зазвичай називають взаємною кореляцією в математичній літературі. Згортка ж математиці включає перевертання вхідного зображення по вертикалі та горизонталі перед виконанням взаємної кореляції. Але в області комп'ютерного зору згортка відноситься до операції з сумою елементів без перевертання.

2.3.2 Архітектура згорткової мережі

Згорткова мережа складається з декількох слоїв:

- Згорткового шару (convolutional)
- Підвибіркового або агрегувального шару (subsampling або pooling)
- Повнозв'язних шарів (fully-connected layer)

Основна ідея алгоритму в чергуванні довільного порядку цих шарів, що зображено на рис.2.17.

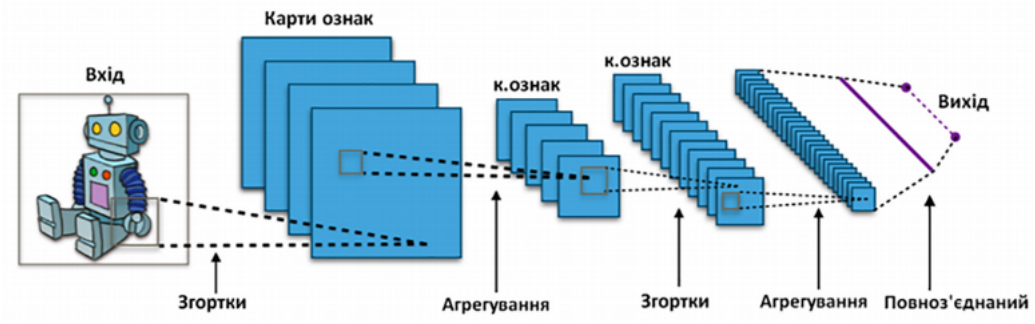


Рисунок 2.17 – Приклад архітектури згорткової нейромережі

2.3.3 Шар згортки

Мета цього шару – виокремити з вхідного зображення високорівневі об’єкти, наприклад краї. Зазвичай, перший шар ConvLayer відповідає за захват низькорівневих функцій (колір, краї, градієнт). В цьому шарі формуються карти ознак, завдяки групуванню нейронів з однаковими вагами. Особливістю є те, що всі нейрони карти ознак пов’язані з частиною нейронів попереднього шару.

На шарі згортки використовуються фільтри та ми можемо їх собі представити як «датчики», які допомагають визначити якусь ознаку на вхідному зображенні.

Виконання згорток замість підключення кожного пікселя до блоків нейронної мережі має дві переваги. По-перше, це зменшує кількість параметрів, яких нейронній мережі треба вивчити. Тобто заміть того, щоб вивчити ваги, що з’єднують пікселі, нам потрібно лише знати вагу фільтра, який зазвичай набагато менше за вхідне зображення. По-друге, це зберігає відносне положення пікселів.

В операції згортки є одна важлива деталь – заповнення (padding). Існує декілька типів заповнень, але розглянемо лише одну – нульову (zero-padding), щоб зрозуміти мету цієї дії.

Коли ми робимо згортку, то зображення буде зменшуватись. А якщо використовувати нейронну мережу з сотнями слоїв, то це приведе до дуже малого зображення на виході. Тож заповнення дозволяє розширити область, в якій

згорткова нейронна мережа обробляє зображення. Ядро – фільтр нейронної мережі, який переміщується по зображенню, сканує кожний піксель та перетворює данні в менший або більший формат. Для того, щоб «допомогти» ядру в обробці зображень, використовують доповнення відступів до зображення. Це дозволяє нейромережі більш точно аналізувати зображення.

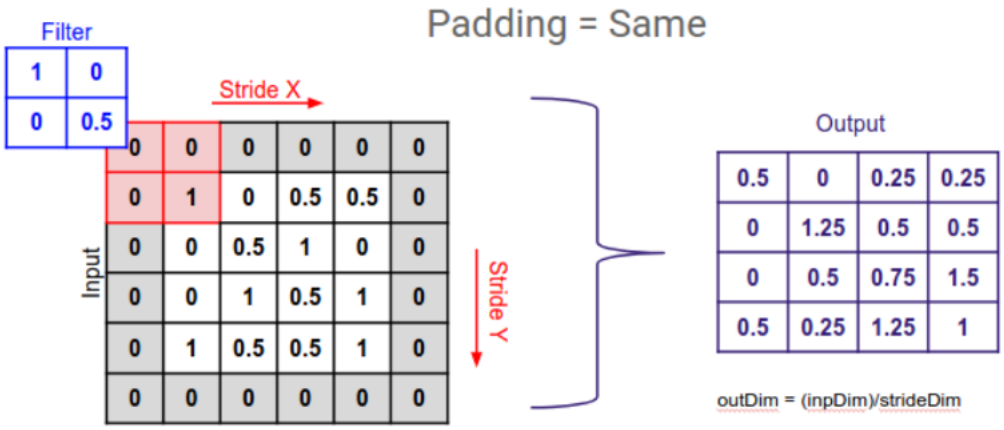


Рисунок 2.18 – Доповнення нулями країв зображення

На малюнку 2.18 можна побачити, що завдяки доповненню нулів, інформації про малюнок стало більше. Тому що фільтр проходить через крайні точки (ненульового) зображення не один раз. Тобто стає більша точність роботи мережі.

Ще один параметр, який можна обирати – крок (stride). Воно показує на кількість пікселів, які наш фільтр буде переміщати кожного разу. Більшість мереж мають крок з одиничним значенням, але якщо збільшити крок, то вихідне зображення буде зменшуватись ще більше. Але для деяких зображень, де значення близьких пікселів дуже схожі, нам не треба аналізувати кожен піксель зображення, так як ми зможемо отримати ту ж інформацію з меншим зображенням.

2.3.4 Підвибірковий шар

Підвибірковий шар використовується для зменшення вхідного об’єму. Також це називають субдискретизацією або понижувальною дискретизацією, яка зменшує розмірність кожної карти, але зберігає важливу інформацію.

Існує два метода: максимальне та середнє об’єднання (рис.2.19). При максимальному об’єднанні максимальні значення зберігаються, а всі інші – відсікаються. При середньому – в пулі зберігається середнє значення всіх значень у вибраному ядрі.

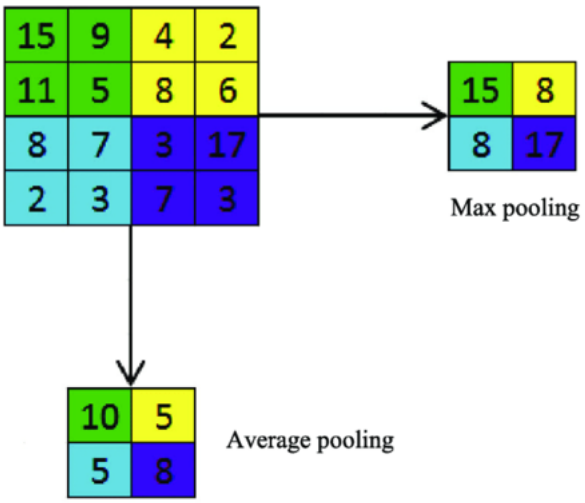


Рисунок 2.19 – Два типи об’єднання

Зазвичай використовується максимальне об’єднання так як воно дає кращі результати.

2.3.5 Повнозв’язний шар

Після шару об’єднання ми групуємо наші дані у вектор та передаємо їх в повнозв’язний шар, як у звичайній нейронній мережі. Цей шар існує для додання нелінійності до наших даних та класифікації складних елементів, що були виокремлені з попередніх шарів.

Щоб передати вхідне зображення до даного шару, необхідно вирівняти зображення так, щоб всі значення пікселів були записані в одному стовпці. Всі ваги ініціалізуються невеликим випадковим числом та навчаються з використанням алгоритму зворотного поширення.

Кінцевий результат розраховується завдяки функції активації (наприклад, softmax), яка розраховує ймовірності кожного класу для заданих ознак.

2.4 Генеративні змагальні нейронні мережі

Змагальні нейронні мережі (GAN) – клас алгоритмів штучного інтелекту, що використовується в машинному навчанні без вчителя.

Дані мережі представлять архітектуру глибоких нейронних мереж, які складаються з двох мереж: генератора (G) та дискримінатора (D). Основна ідея в «змаганні» дискримінатора та генератора.

Принцип алгоритму в тому, що G генерує певні зразки (наприклад, зображення або відео), а мережа D повинна вирішити чи є цей зразок істинним, чи згенерованим. Тобто генератор повинен створювати такі зразки, які б дискримінатор відніс до істини. А дискримінатор навпаки повинен забракувати згенеровані зразки. Таким чином між дискримінатором та генератором виникає антагоністична гра, в результаті якої обидві нейронні мережі навчаються без вчителя.

SRGAN (Super Resolution Generative Adversarial Network) – генеративна змагальна мережа, що створена для підвищення роздільної здатності зображень.

$$\min \max E_{I^{HR}\sim P_{train}}\left[\log D_{\theta_D}(I^{HR})\right] + E_{I^{LR}\sim P_G}[\log(1 - D_{\theta_D}\left(G_{\theta_G}(I^{LR})\right)] \qquad (2.8)$$

2.4.1 Архітектура генеративної змагальної мережі

Дискримінатор та генератор навчаються одночасно та як тільки генератор буде навчений, він буде знати достатньо про розподіл навчальної вибірки. Тому зможе генерувати нові вибірки, які будуть мати схожі властивості.

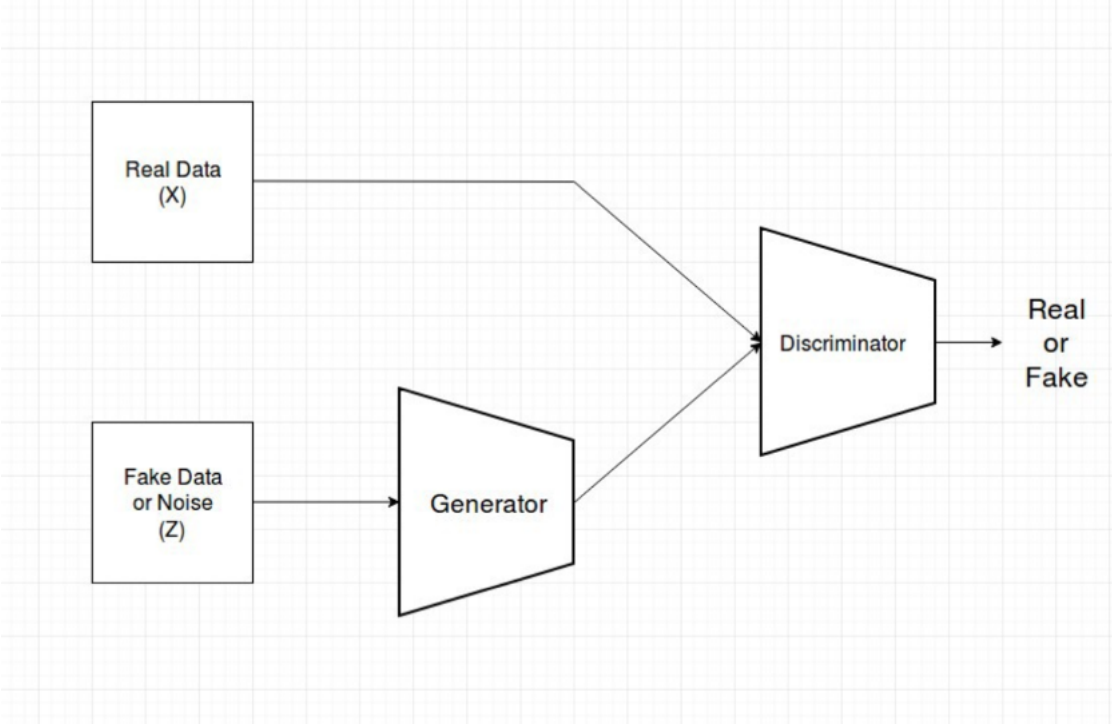


Рисунок 2.20 – Базова архітектура генеративної змагальної мережі

Процедура навчання мережі:

- Спочатку обробляємо зображення високої роздільної здатності (High resolution або HR), щоб отримати зображення низької роздільної здатності (low resolution або LR) з пониженою частотою дискретизації. Тепер маємо образи HR та LR для набору тренувальних даних.

- Пропускаємо зображення LR через генератор, який збільшить роздільну здатність та отримаємо зображення SR (Super Resolution).

- Використаємо дискримінатор для розрізнення зображень HR та SR. Також дискримінатор поширить втрати GAN, щоб навчити дискримінатор та генератор.

Розглянемо схему мережі для генератора та дискримінатора на рис. 2.21.

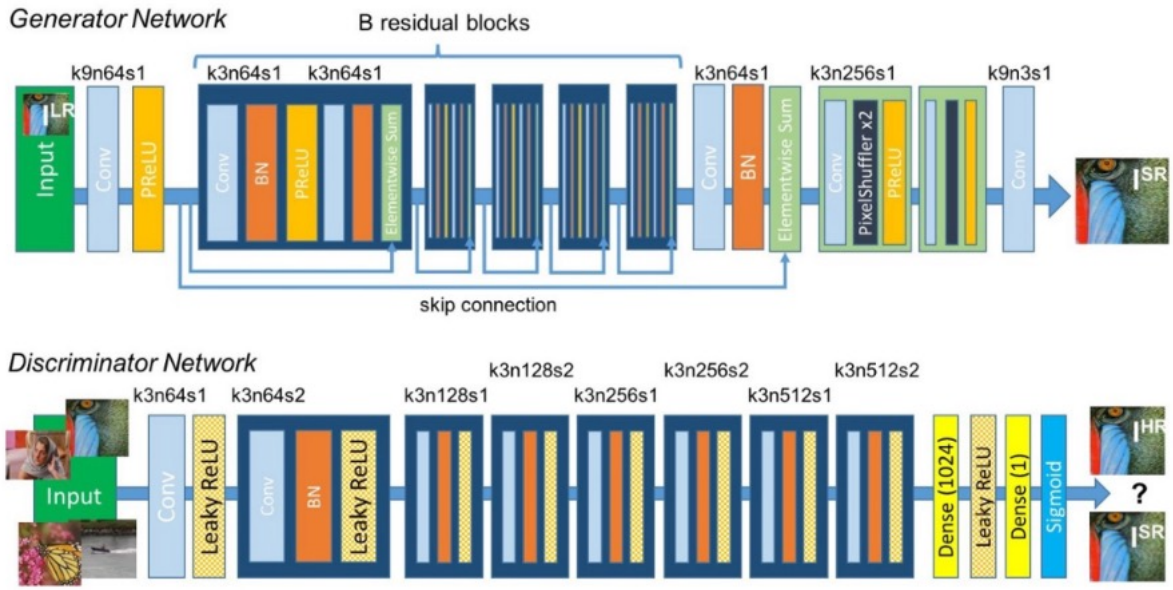


Рис. 2.21 – Схема структури мереж генератора та дискримінатора

По більшій частині вони складаються зі згорток, пакетної нормалізації та ReLU. Residual blocks або ж залишкові блоки використовуються тому, що вони полегшують процес навчання та дозволяє мережам бути значно глибшими, що призводить до підвищення продуктивності. Тож 16 залишкових блоків використовуються в генераторі.

2.4.2 Функції втрат

Визначення функції втрат є важливим аспектом у створенні та ефективній роботі генеративної змагальної мережі. Є два типи функцій втрат, значення яких ми будемо вираховувати.

Перша – перцептивна (perceptual) або функція сприйняття, що обчислюється як зважена сума функції втрат змісту (контенту) та функції втрати змагальності. Функцію сприйняття обчислюють для того, щоб змусити мережу віддати переваги справжньому зображенню, а не згенерованому, з метою ввести в оману дискримінатор. Формула перцептивної функції втрат або втрати сприйняття (2.9) має вигляд:

$$l^{SR} = l_X^{SR} + 10^{-3} l_{Gen}^{SR} \quad (2.9)$$

Друга – функція втрати змагальності. Вона обчислюється на основі ймовірності того, чи є згенероване зображення є HR зображенням.

Втрату змагальності можна обрахувати за формулою 2.10:

$$l_{Gen}^{SR} = \sum_{n=1}^{N} -\log D_{\theta_D}(G_{\theta_G}(I^{LR})) \quad (2.10)$$

Втрату контенту використовують, щоб зберегти схожість сприйняття замість подібності по пікселям. Функція визначається на основі шарів активації ReLU попередньо протренованої VGG мережі та обчислюється як евклідова відстань між ознаками початкового зображення та отриманого в результаті генерування. Крім втрат на основі VGG мережі, функція може бути розрахована як втрата MSE.

Втрата MSE порівнює HR зображення із згенерованим зображенням:

$$l_{MSE}^{SR} = \frac{1}{r^2 WH} \sum_{x=1}^{rW} \sum_{y=1}^{rH} (I_{x,y}^{HR} - G_{\theta_G}(I^{LR})_{x,y})^2 \quad (2.11)$$

$$l_{VGG}^{SR} = \frac{1}{W_{i,j} H_{i,j}} \sum_{x=1}^{W_{i,j}} \sum_{y=1}^{H_{i,j}} (\phi_{i,j}(I^{HR})_{x,y} - \phi_{i,j}(I^{SR})_{x,y})^2 \quad (2.12)$$

2.4.3 PSNR

PSNR (peak signal-to-noise ratio) або пікове співвідношення сигналу до шуму є співвідношенням між найбільшим значенням сигналу та потужності шуму. Це значення використовується для вимірювання рівня спотворень зображення при стисненні.

$$PSNR = 10log_{10}\left(\frac{MAX_1^2}{MSE}\right) = 20log_{10}(\frac{MAX_1}{\sqrt{MSE}}) \quad (2.9)$$

де MSE – середньоквадратичне відхилення, що обчислюється за формулою:

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}|I(i,j) - K(i,j)|^2 \quad (2.10)$$

2.5 Помилки при створенні генеративної змагальної мережі

1) Потрібно використовувати більші ядра та фільтри. Більші ядра покривають більшу кількість пікселів в попередньому зображені шару і тому можуть сприймати більше інформації. Використання ядер 3x3 в дискримінаторі призвело до швидкого наближенню втрати дискримінатора до 0.

2) Додавання пакетної нормалізації покращує результат.

3) Не зупиняти тренування на ранніх етапах. Помилка в тому, що часто тренування переривають після декількох міні-партій тренування, коли бачимо, що втрати не досягли прогресу, або якщо згенеровані зображення залишалися шумними. А переривають тому, що думають, що зекономлять час. Але потрібно пам’ятати, що GAN займає досить багато часу для навчання та ініціалізує декілька значень втрат та іноді вибірки не показують прогресу. Тому інколи варто трішки більше почекати на результат. Але якщо втрати дискримінатора швидко наближаються до 0, тоді скоріш за все краще почати навчання спочатку.

2.6 Висновки до розділу

У цьому розділі було розглянуто різні нелінійні інтерполяційні методи такі як: метод ближнього сусіда, білінійний та бікубічний метод, фільтр Ланцоша. Проаналізовані методи використовують різну кількість пікселів для інтерполяції, чим відрізняються один від одного. Найбільш гарний результат отримуємо методом бікубічної інтерполяції та за допомогою фільтру Ланцоша, так як вони використовують більше пікселів навколо невідомого, за інших. Але попри це, результуюче зображення не є чітким, різкі границі втрачаються та зображення стає досить розмитим. Тож дані інтерполяційні методи можна застосовувати, але треба розуміти, що дуже гарного та якісного результату вони не дадуть.

Було проаналізовано структуру звичайної нейронної мережі, згорткової та генеративної змагальної мережі. Розглянуті основні структурні елементи та особливості кожної мережі. Так, основними в згортковій мережі є саме шари згортки, а в генеративній змагальній мережі основними елементами є генератор та дискримінатор, що взаємодіють між собою та утворюють мінімаксну гру, де для генератора важливо згенерувати таке зображення, щоб дискримінатор зарахував його за істинне. Дана мережа також має важливу структурну особливість – функцію втрат, яка складається з функції втрати контенту та функції втрати змагальності. Саме вони потрібні для ефективного навчання та роботи системи.

Було проведено аналіз помилок, які найчастіше зустрічаються у схожих роботах, які допомогли створити власну систему без цих прикладів.

3 ОПИС ОБРАНИХ ТЕХНОЛОГІЙ

3.1 Вибір середи розробки

Мовою програмування даної системи є Python. Вона є найпоширенішою для задач машинного навчання, обробки та класифікації даних. Існує велика кількість бібліотек, які допомагають розробникам краще створити продукт.

Середою програмування є Google Colaboratory, де файлом, в якому працює розробник є Jupyter Notebook. Платформа дозволяє виконувати різні ділянки коду, що полегшує роботу. Файли твій код буде зберігатись у цьому файлі та знаходитиметься на Google drive. Цей сервіс надає безкоштовний GPU, але з обмеженою пам’яттю. Середою підтримується Python 3.6, на якому і була написана система.

Особливістю є те, що у користувача обмежений час користування сервісом. Є всього 12 годин, щоб натренувати свою систему, потім runtime буде зупинено та всі дані, які користувач не встиг зберегти – зникнуть. Так, у даній роботі, зберігались всі чек-поінти на гугл-диск кожного разу, коли система тренувалась. Щоб у разі переривання не зникли зовсім всі дані.

Дана середа програмування буда вибрана тому, що має більшу потужність, ніж стандартний комп’ютер.

Основні бібліотеки, що були використані при створенні системи: NumPy, Keras, Tensorflow, openCV.

3.2 Вибір тренувальної вибірки та архітектура мережі

Обрати тренувальний набір даних не дуже складно, їх є досить багато у відкритому доступі. Але важливим моментом є те, щоб у вибірці були зображення як малої роздільної здатності (low resolution), так і великої (high resolution).

У роботі був вибраний датасет DIV2K. У цьому наборі даних є 800 зображень низької та ще 800 високої роздільної здатності для тренування системи. Також 100 зображень для перевірки роботи системи, та ще 100 зображень для тестування.

Архітектуру генеративної змагальної мережі була візуально представлена на рис. 2.22.

По-перше, треба відзначити, що наша мережа складається з трьох мереж. Спочатку, нам треба навчити генератор генерувати значення пікселів. Це робиться за допомогою згорткової мережі. Вже потім, коли наш генератор навчений обробляти зображення та генерувати значення, ми запускаємо навчатись генератор та дискримінатор разом. Саме тоді генератор вчиться робити такі значення, щоб вони були ближче до істини. Так, щоб ці значення дискримінатор пропустив як істину. В свою чергу дискримінатор навчається відрізняти згенероване зображення від істини. В основі генеративної мережі лежить 8 залишкових блоків (residual blocks) з однаковим внутрішнім наповненням. В кожному блоці знаходиться два шари згортки з ядрами 3x3 та 64 картами ознак, після яких є етап пакетної нормалізації. В якості функції активації використовується функція PReLU. Тобто проходячи через цей етап, зображення збільшується попіксельно, проходячи два шари згортки. Для навчання дискримінатора відрізняти SR зображення від HR, використовувалась функція активації LeakyReLU. Дискримінаторна мережа містить в собі 8 шарів згортки та має таку кількість ядер фільтру 3x3, яка збільшується кожного разу в 2 рази (від 64 до 512). Отримані 512 карт ознак проходять через кінцеву функцію активацію – сигмоїду, для отримання значення ймовірності класифікації об’єкта.

Не можна запускати генератор та дискримінатор навчатися одночасно без попереднього тренування генератора, бо тоді навчання буде не ефективним та його майже не буде відбуватись.

Метод оптимізації, який був використаний в роботі – Адам з learning rate (параметром навчання) 0.0001. Кількість епох з перед-тренуванням генератора – 30000. Кількість епох з навчанням генератора та дискримінатора одночасно – 20000.

3.3 Аналіз та результати роботи

Розглянемо графіки функції втрат генератора та дискримінатора на рис.3.1-3.6.

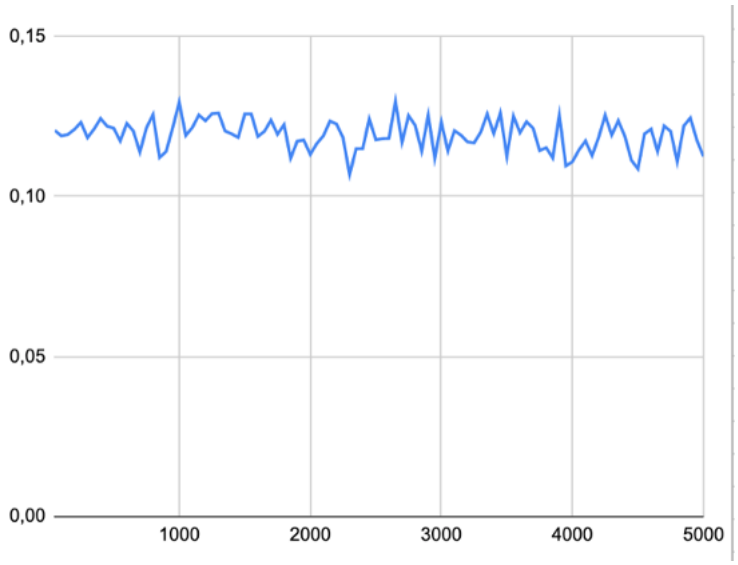


Рисунок 3.1 - Графік функції втрат генератора за 5000 кроків

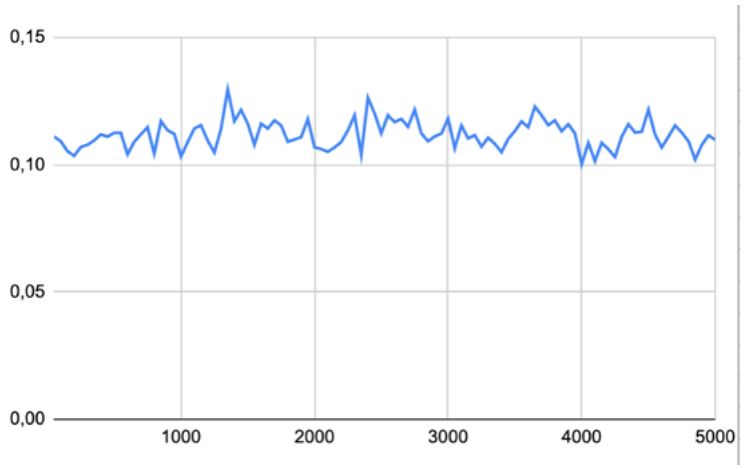


Рисунок 3.2 - Графік функції втрат генератора за другі 5000 кроків

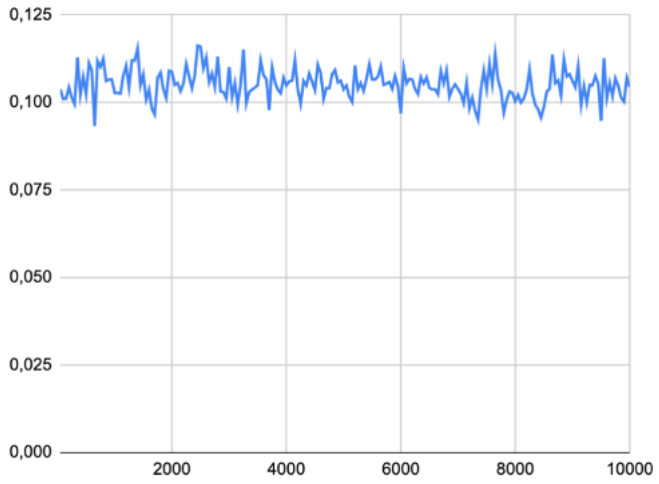


Рисунок 3.3 - Графік втрат генератора на останніх 10000 кроків

Спираючись на дані графіки можна зробити висновок, що кожного разу функція втрат стає меншою. Досить добре, що вона має різні значення в невеликих межах, бо це свідчить про те, що генератор навчається. Також треба відмітити те, що це графіки саме з другої частини тренування генератора, коли він вже тренується разом з дискримінатором. Через це, немає сильних перепадів та різкого спуску функції втрат.

Схожі графіки, але більш виражені ми можемо побачити на графіках втрат дискримінатора.

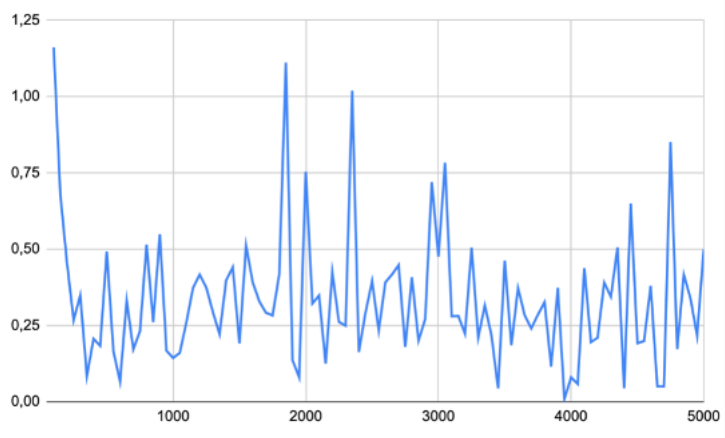


Рисунок 3.4 - Графік функції втрат дискримінатора за перші 5000 кроків

Рисунок 3.5 - Графік функції втрат дискримінатора за другі 5000 кроків

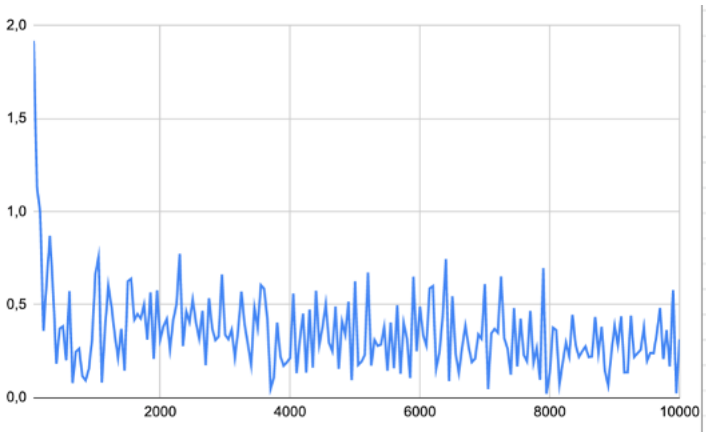


Рисунок 3.6 - Графік функції втрат дискримінатора за останні 10000 кроків

Отже, за графіками навчання генератора можна сказати, що навчання ніколи не зупинялось, про що свідчать перепади значень функції втрат. Також можна зазначити, що функція втрат зменшувалася з кожним тренуванням системи.

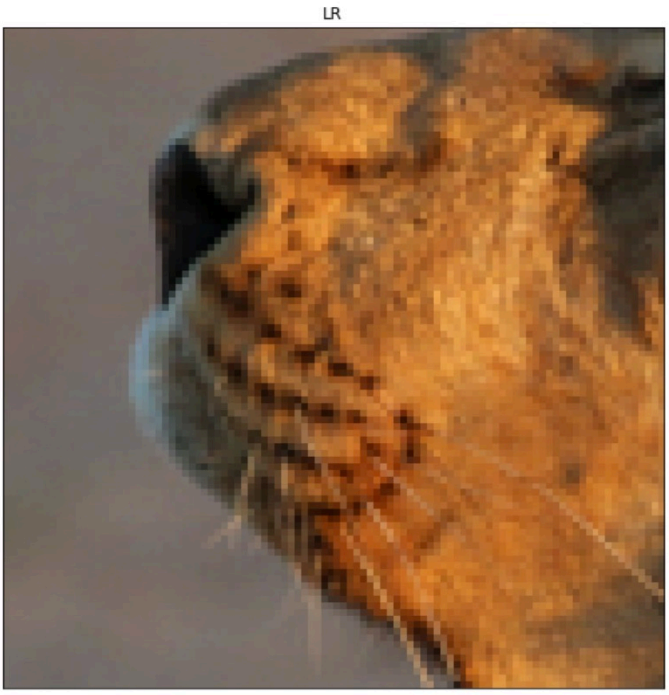


Рисунок 3.7 - вхідне зображення

Далі тренувалась генеративна мережа. Розглянемо результати роботи тільки згорткової нейронної мережі. Різниця між зображеннями в 10000 кроків навчання (зліва – перша ітерація, справа – друга, нижче - остання).

Рисунок 3.8 - Порівняння роботи згорткової мережі для генератора

Далі – результати роботи саме генеративної змагальної мережі. Різниця між зображеннями (зліва, справа та знизу) : 5000, 5000 та 10000 кроків відповідно.

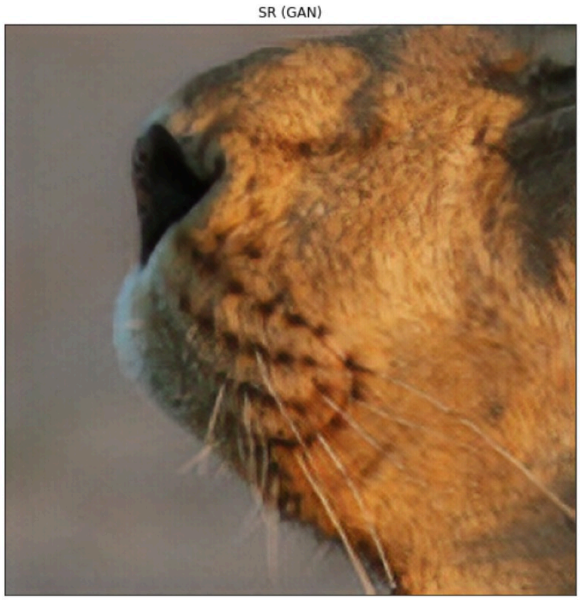


Рисунок 3.9 - Порівняння роботи генеративної змагальної мережі

Спочатку, на перших 5000 кроках, зображення було з відтінками сірого та не мало чітких меж. Але на після останнього треенування системи, можна побачити, що зображення стало більш деталізованим. Шерсть у тварини більш виражена та не є такою розмитою, як, наприклад, у згортковій мережі на рис 3.8. Кольори також стають не такими сірими та не надмірно насиченими. Тож генеративна змагальна мережа показала гарний результат.

Розглянемо ще один приклад.

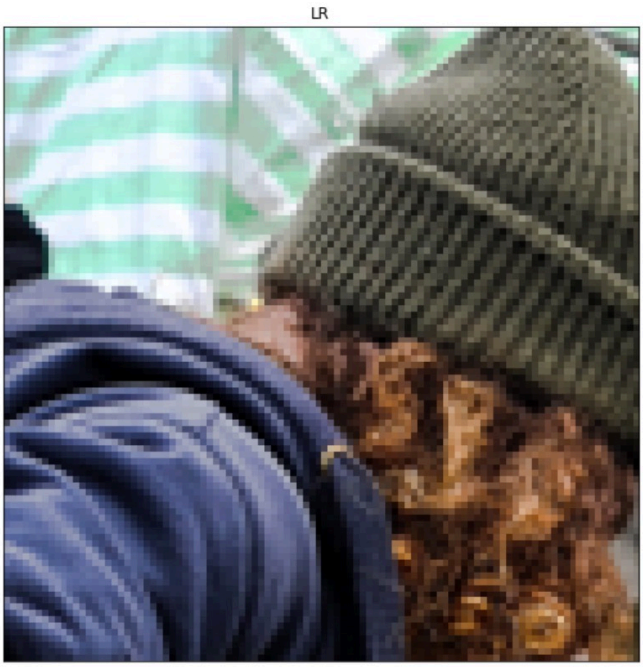


Рисунок 3.10 - Вхідне зображення

Рисунок 3.11 - Приклад роботи згорткової мережі

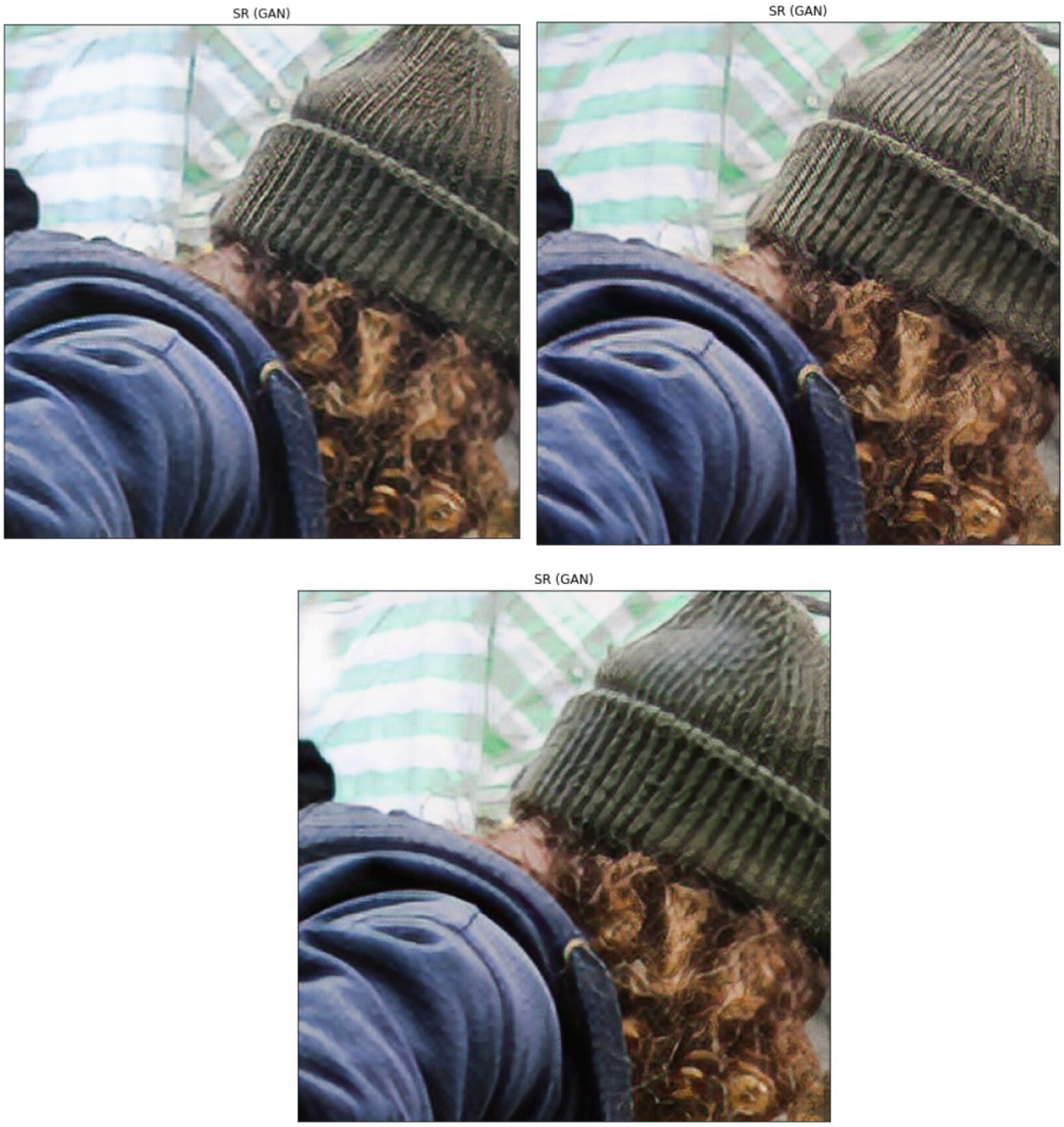


Рисунок 3.12 - приклад роботи генеративної змагальної мережі

На цьому прикладі можна побачити, що генеративна змагальна мережа набагато краще працює з різними текстурами Також, вона не робить зображення розмитим, або нечітким.

3.4 Висновки до розділу

Було реалізовано систему покращення якості зображення за допомогою генеративної змагальної мережі. Для створення даної системи потрібно спочатку натренувати генератор, а потім розпочати навчання як генератора, так і дискримінатора разом. За результатами, які було отримано, можна сказати, що генеративна змагальна мережа працює краще, то по-перше, вона не викривляє кольори, контури зображення не стають дуже розмитими. Було досліджено, що зі збільшенням епох, мережа краще працює з різними текстурами та кольорами.

Також було отримано графіки функцій втрат, яка з кожним тренуванням системи зменшувались. Це свідчить про те, що система працює більш чітко та точно.

Отримані результати роботи показали, що генеративна змагальна мережа обробляє зображення значно краще, ніж згорткова нейронна мережа. Тож якщо протренувати ще більше дану систему, вона буде працювати з більшою точністю та може бути використана для задач відновлення якості зображення без втрати змісту.

4 ПРОЕКТУВАННЯ СИСТЕМИ ПОКРАЩЕННЯ РОЗДІЛЬНОЇ ЗДАТНОСТІ ЗОБРАЖЕНЬ

4.1 Постановка задачі проектування

Проектований програмний продукт є системою покращення роздільної здатності зображення. Система дозволяє завантажувати файли, обробляти їх та виводити результати одразу на екран. Продукт може використовуватись на будь-якій операційній системі, але в тих середовищах, які підтримують мову Python.

4.2 Обґрунтування функцій та параметрів програмного продукту

Виходячи з конкретних цілей, які реалізуються:

F_1 – вибір мови програмування: а) Python, б) C++.

F_2 – вибір фреймворку для розробки нейронної мережі: а) Tensorflow, б) Caffe

F_3 – вибір середовища розробки: а) JupyterNotebook, б) VisualStudio.

Виходячи з представлених варіантів будуємо морфологічну карту (рис. 4.1)

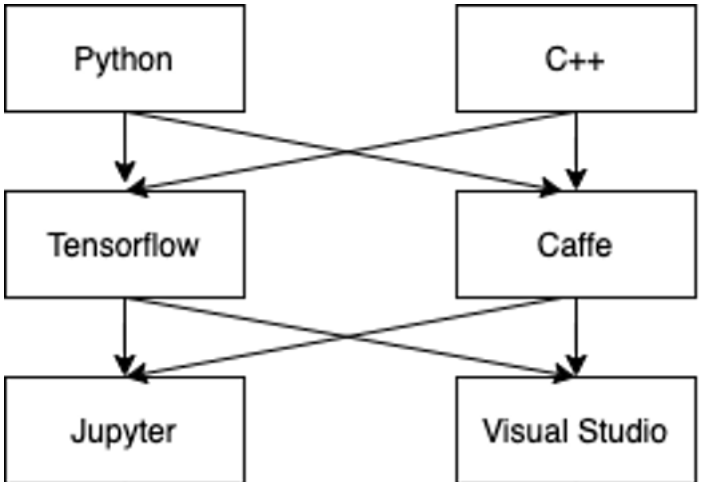


Рисунок 4.1 – Морфологічна карта

Дана морфологічна карта відображає всі можливі комбінації варіантів реалізації функцій, які складають повну множину варіантів програмного продукту.

Спираючись на морфологічну карту, була побудована позитивно-негативна матриця (табл. 4.1)

Таблиця 4.1 - Позитивно-негативна матриця

Основні функції	Варіанти реалізації	Переваги	Недоліки
F1	a	Кросплатформність, займає менше часу при написанні коду, багато готових бібліотек	Низька швидкодія
	b	Має високу швидкодію, можна оптимізувати пам’ять вручну	Займає багато часу при розробці коду
F2	a	Безкоштовний фреймворк, зрозуміла документація	Відсутність нативної реалізації Windows
	b	Швидкість, безкоштовність	Необхідність додаткового завантаження, погано оформлена документація
F3	a	Можна виконувати на віртуальному комп’ютері та не задіювати власні потужності, виконує окремі ділянки коду	Повільний
	b	Високий рівень підтримки, багато додаткових інструментів	Треба додатково завантажувати, займає багато пам’яті

Для характеристики прототипу програмного додатку використовуємо параметри X1 – X4. На основі даних, що представлені у літературі, визначаємо мінімальні, середні отримуванні та максимально допустимі значення(табл. 4.2)

Таблиця 4.2 – Основні параметри ПП

Назва параметра	Умовні позначення	Одинці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X1	Оп/мс	19000	11000	2000
Об’єм коду	X2	К-ть строк	2000	1500	1000
Час обробки запитів користувача	X3	Мс	500	250	50
Об’єм пам’яті для збереження даних	X4	Мб	32	16	8

За даними, наведеними у таблиці 4.2, будуються графічні характеристики параметрів – рисунки 4.2 – 4.5.

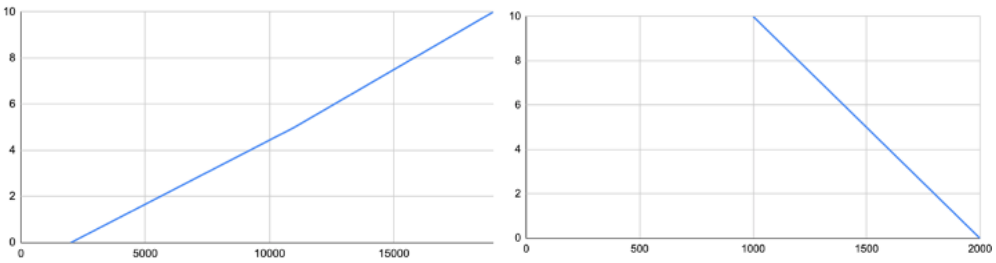


Рисунок 4.2 – Швидкодія мови програмування та потенційний об’єм коду

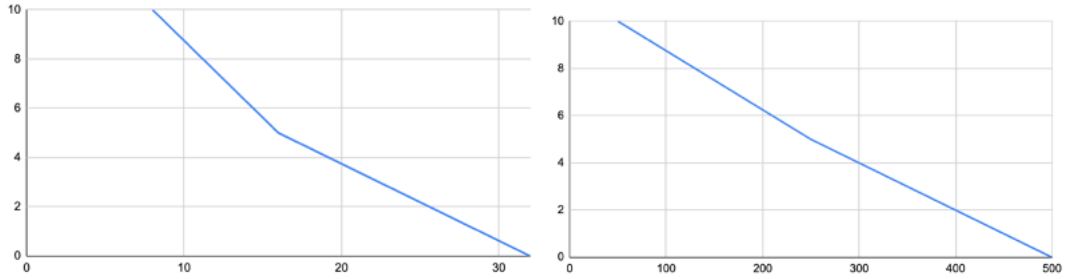


Рисунок 4.3 – Об’єм пам’яті збереження даних та час обробки даних

Вагомість параметрів оцінюється за допомогою методів попарного зрівняння. Рани варіюються від 1 до 4 (вищий – нижчий). Результати наведені в табл. 4.3.

Таблиця 4.3 – Результати ранжування параметрів

параметр	Ранг параметра за оцінкою експерта							Сума рангів в R_i	Відхилення Δi	Δ_{i^2}
	1	2	3	4	5	6	7			
X1	2	3	4	3	1	3	3	19	1,5	2,25
X2	4	4	3	4	4	4	4	27	9,5	90,25
X3	1	1	1	2	2	1	1	9	-8,5	72,25
X4	3	2	2	1	3	2	2	15	-2,5	6,25
Разом	10	10	10	10	10	10	10	70	0	171

Таблиця 4.4 - Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	>	>	<	>	>	>	>	>	1.5
X1 і X3	<	<	<	<	>	<	<	<	0.5
X1 і X4	>	<	<	<	>	<	<	<	0.5
X2 і X3	<	<	<	<	<	<	<	<	0.5
X2 і X4	<	<	<	<	<	<	<	<	0.5

X3 і X4	>	>	>	<	>	>	>	>	1.5

Визначимо коефіцієнт конкординації:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 * 171}{49 * (4^3 - 4)} = 0{,}98 > W_k = 0{,}67$$

Так як коефіцієнт конкординації більше нормативного, результати вважають достовірними.

Розрахунок вагомості параметрів наведено в табл. 4.5.

Таблиця 4.5 розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітерація		Друга ітерація		Третя ітерація	
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	b_i^1	$\text{К}_{\text{ві}}^1$	b_i^2	$\text{К}_{\text{ві}}^2$
X1	1	1.5	0.5	0.5	3.5	0.219	12.25	0.2	48.25	0.211
X2	0.5	1	0.5	0.5	2.5	0.156	11.5	0.188	36.375	0.159
X3	1.5	1.5	1	1.5	5.5	0.344	21.25	0.347	81.25	0.356
X4	1.5	1.5	0.5	1	4.5	0.281	16.25	0.265	62.5	0.247
Разом:					16	1	61.25	1	228.375	1

Враховуючи дані з порівнянь варіантів реалізацій функцій можна виключити з реалізацій функцій наступні варіанти: F1(б), F3(б)

Залишаються наступні варіанти:

1) F1а-F2а-F3а
2) F1а-F2б-F3а

Таблиця 4.6 Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості

F1(X1)	А	11000	5	0.274	1.37
F2(X2)	А	1600	4	0.356	1.42
F2 (X2)	Б	1800	2	0.356	0..71
F2 (X3)	А, Б	300	4	0.159	0.64
F3 (X4)	А,Б	16	5	0.211	1.1

За цими даними визначаємо рівень якості кожного з варіантів:

$$K_{я1} = 1.37 + 0.64 + 1.42 + 1.1 = 4.53$$

$$K_{я2} = 1.37 + 0.64 + 0.71 + 1.1 = 3.72$$

Як видно з розрахунків, кращим є перший варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.3 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Програмування продукту на мові Python;
2. Створення та обробка коду в JupyterNotebook;

Кожний з варіантів має додаткове завдання, які є реалізаціями розгалужених варіантів розробки незалежного модуля. Далі наведено варіанти додаткових завдань:

3.1 Використання фреймфорку Tensorflow

3.2 Використання фреймворку Caffe

У варіанті 1 присутнє наступне додаткове завдання під номером 3.1.

У варіанті 2 присутнє наступне додаткове завдання під номером 3.2.

За ступенем новизни завдання 1 відноситься до групи А, завдання 2, 3.1відноситься до групи Б, завдання 3.1 відноситься до групи В.

За складністю алгоритми, які використовуються в завданні 1 належать до 1 групи, завдання 2, 3.1, 3.2 – до 3 групи.

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $T_р$ = 90 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $К_п$ = 1.7. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $К_{ск}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{ст}$ = 0.8. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 90 * 1.7 * 0.8 = 122.4 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для інших завдань. Для другого завдання (використовується алгоритм третьої групи складності , ступінь новизни Б), тобто $T_р$ =19 людино-днів, $К_п$=0.9, $К_{ск}$ = 1, $К_{ст}$ = 0.8. Отже:

$$T_2 = 27 * 0.9 * 0.8 = 19.44 \text{ людино-днів.}$$

Аналогічно для завдання 3.1 та 3.2:

$$T_3 = 27 * 0.9 * 0.8 = 19.44$$

$T_р$=12 людино-днів, $К_п$ =0,6, $К_{ск}$ =1, $К_{ст}$ =0,8

$$T_4 = 12 * 0.6 * 0.8 = 5,76$$

Визначимо повну трудомісткість варіантів (людино-днів):

$$T_1 = 122,4 + 19,44 \ + 19,44 = 161,28$$

$$T_2 = 122,4 + 19,44 + 5,76 = 147,6$$

Найбільш трудомістким завданням є 1, найбільш трудомістким варіантом є 1. Далі вважається, що робочий день складає 8 годин, в тижні п’ять робочих днів. В розробці бере участь два програміста з окладом 7000 грн та аналітик з окладом 9500 грн. Визначимо середню заробітну плату за годину:

$$С_ч = \frac{2 * 7000 + 9500}{3 * 21 * 8} = 46,62$$

Заробітна розробників за варіантами становить:

$$С_{ЗП} = 46{,}62 * 8 * 161{,}28 = 60150{,}9$$

$$С_{ЗП} = 46{,}62 * 8 * 147{,}6 = 55048{,}9$$

Відрахування на єдиний соціальний внесок становить 22%:

$$С_{ВІД} = 60150{,}9 * 0{,}22 = 13233{,}19$$

$$С_{ВІД} = 55048{,}9 * 0{,}22 = 12110{,}75$$

Визначимо витрати на оплату однієї машино-години. Так як одна машина обслуговує двох програмістів з окладом 7000 грн. з коефіцієнтом зайнятості 0.6 та одного аналітика з окладом 9500, то для трьох машин отримаємо:

$$С_г = 12 * 7000 * 2 * 0{,}6 + 12 * 9500 * 0{,}6 = 169\,200$$

Враховуючи додаткову заробітну плату 40%

$$С_{ЗП} = 169200 * (1 + 0{,}4) = 236\,880 \text{ (грн)}$$

Відрахування на соціальне страхування 22%

$$С_{ВІД} = 236880 * 0{,}22 = 52\,113{,}6$$

Розрахуємо амортизаційні підрахунки (амортизація 25%, вартість ЕОМ 21000 грн)

$$С_А = К_{ТМ} * К_А * Ц_{ПР} = 1{,}15 * 0{,}25 * 21000 = 6037{,}5 \text{ (грн)}$$

Розрахуємо витрати на ремонт та профілактику:

$$С_Р = К_{ТМ} * Ц_{ПР} * К_Р = 1{,}15 * 21000 * 0{,}05 = 1207{,}5 \text{ (грн)}$$

Розрахуємо ефективний годинний фонд часу ПК за рік:

$$Т_{ЕФ} = (365 - 142 - 16) * 8 * 0{,}8 = 1324{,}8 \text{ год}$$

Розрахуємо витрати на електроенергію

$$С_{ЕЛ} = 1324{,}8 * 0{,}6 * 0{,}8 * 1{,}75 = 1112{,}83 \text{ грн}$$

Накладні витрати рівні:

$$С_Н = 21000 * 0{,}67 = 14\,070 \text{ (грн)}$$

Отже експлуатаційні витрати:

$$С_{ЕКС} = 236\,880 + 52\,113{,}6 + 6037{,}5 + 1207{,}5 + 1112{,}83 + 14\,070 = 311\,421{,}43$$

Собівартість однієї машино-години буде:

$$\text{См-г} = \frac{311\,421{,}43}{1324{,}8} = = 235{,}07 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов’язані з розробкою програмного продукту ведуться на ЕОМ, то витрати на оплату машинного часту в залежності від обраного варіанту, складають:

$$См = 235,07 * 8 * 161,28 = 303296,71 \text{ (грн.)}$$

$$См = 235,07 * 8 * 147,6 = 277570,65 \text{ (грн.)}$$

Накладні витрати складають 67% від заробітної плати:

$$С_Н = 303296,71 * 0,67 = 203208,79 \text{ (грн)}$$

$$С_Н = 277570,65 * 0,67 = 185972,33 \text{ (грн)}$$

Отже, вартість розробки програмного продукту за варіантами становить:

$$С_{ПП} = 60150,9 + 13233,19 + 303296,71 + 203208,79 = 579889,59 \text{ (грн)}$$

$$С_{ПП} = 55048,9 + 12110,75 + 277570,65 + 185972,33 = 530702,63 \textit{ (грн)}$$

4.4 Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня:

$$К_{ТЕР1} = \frac{4.53}{579889,59} = 7,81 * 10^{-6}$$

$$К_{ТЕР2} = \frac{3.72}{530702,63} = 7,01 * 10^{-6}$$

4.5 Висновок до розділу

Отже враховуючи всі дослідження, що описані вище, можна сказати, що 1 варіант реалізації є найбільш оптимальним зі сторони якісно-економічної оцінки. Його коефіцієнт техніко-економічного рівня складає $7,81 * 10^{-6}$.

Цей варіант реалізації програмного продукту має такі параметри:

– мова програмування – Python;

– фреймворк машинного навчання - Tensorflow;

– середовище розробки – JupyterNotebook.

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, функціонал і швидкодію.

ВИСНОВКИ

Задача масштабування зображень є актуальною для виробників фото та відео матеріалів. Гарне програмне забезпечення може допомогти виробникам у задоволенні потреб користувачів та буде потребувати не таких великих капіталовкладень.

На сам перед було проаналізовано теоретичну базу відносно масштабування зображень. Були розглянуті як інтерполяційні методи покращення роздільної здатності, так і методи на основі нейронних мереж.

Розглянуто структуру згорткової та генеративної змагальної мережі. Виявлені всі структурні елементи та описано процес навчання мережі.

Дана робота була виконана у новому середовищі Google Colaboratory, що дає змогу використовувати безкоштовний GPU. В середньому одне тренування згоркової мережі для тренування генератора на 10000 епох займало біля 10 годин роботи GPU. Таке тренування проводилося тричі. Та ще одне тренування генеративно-змагальної мережі, тобто генератора та дискримінатора, на 5000 епох виконувалося в середньому 6 годин.

Результатом даної роботи є генеративна змагальна мережа, яку досліджували на декількох різних стадіях. Після 5000 епох, ще 5000 епох та після останніх 10000 тренування саме генеративної змагальної мережі. Було проаналізовано вихідні зображення та зауважено тенденцію покращення зображення зі збільшенням кількості ітерацій, що є цілком логічним. Також було проаналізовано функції втрат та їхню зміну при кожному тренуванню системи.

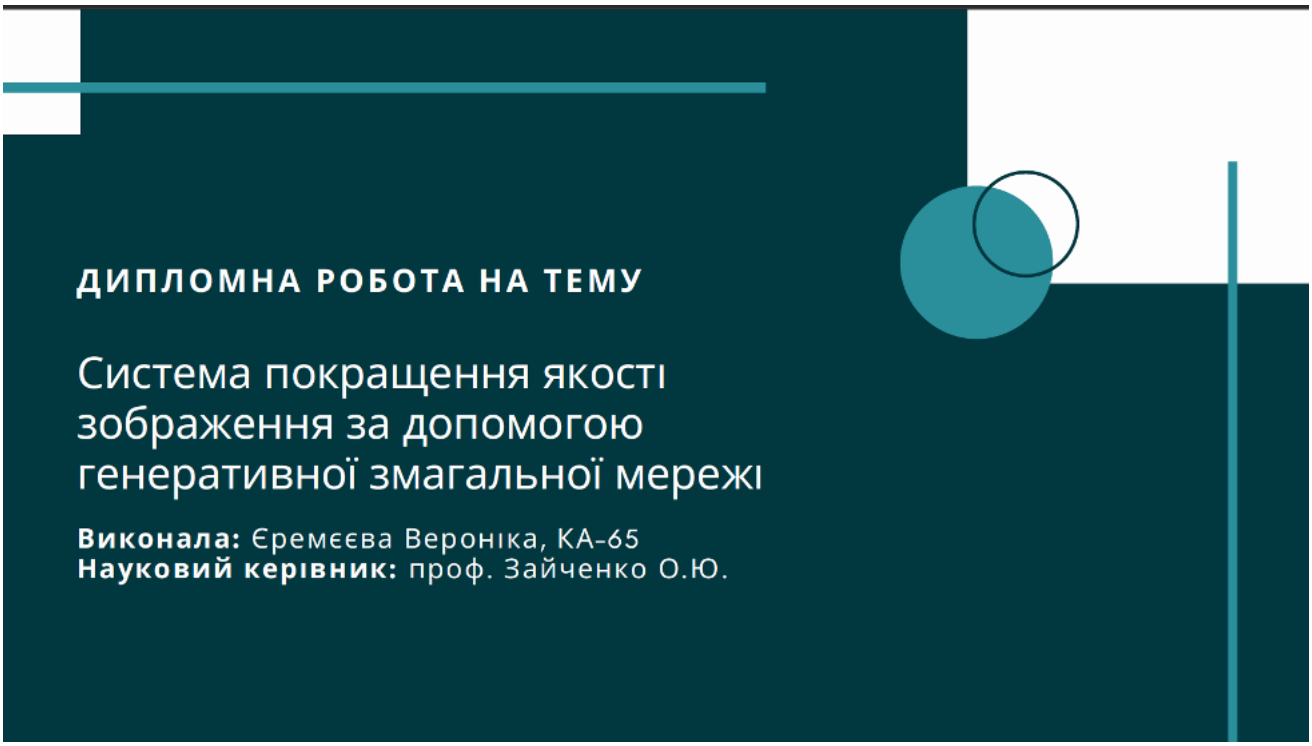
Розроблена система дає змогу покращити вхідне зображення та вивести на екран результати користувачеві.

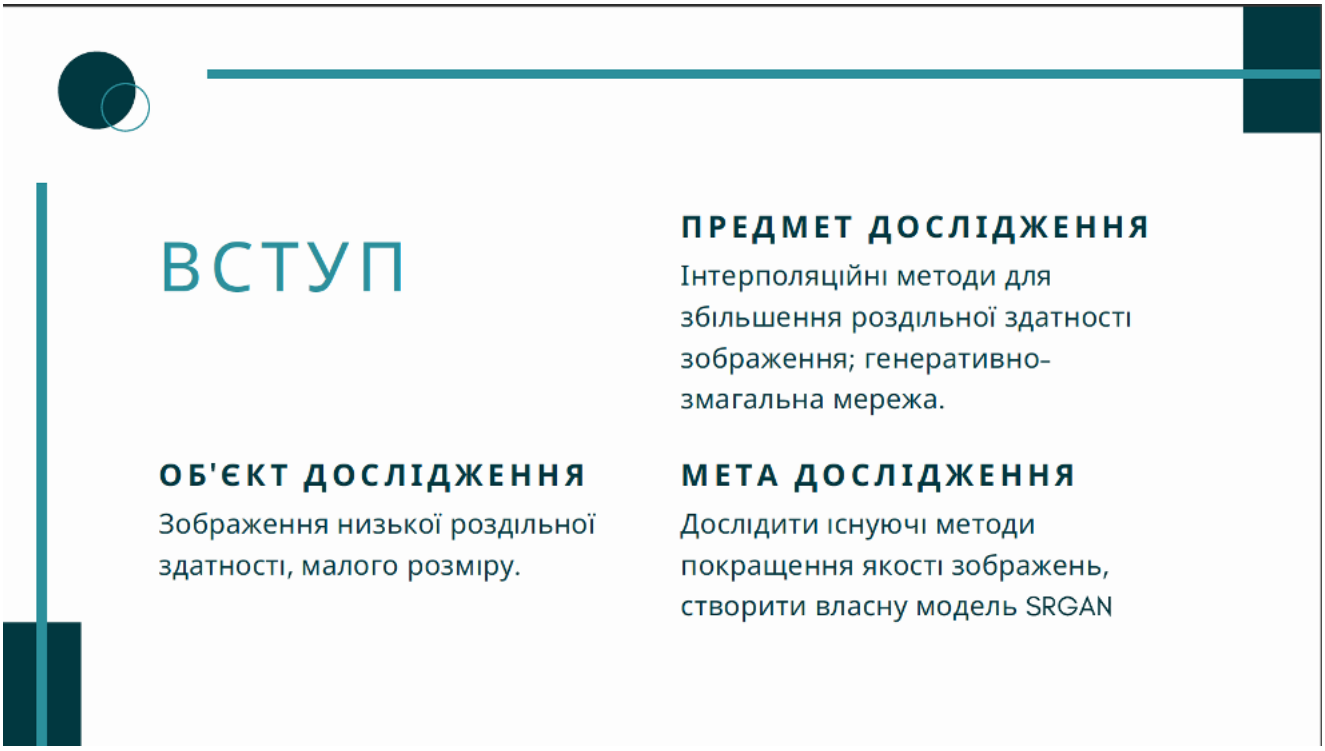
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Нейронные сети. Часть 2. URL: https://habr.com/ru/post/313216/ (дата звернення: 10.06.2020)
2. Методы оптимизации нейронных сетей. URL: https://habr.com/ru/post/318970/ (дата звернення: 10.06.2020)
3. PSNR и SSIM или как работать с изображениями под C. URL: https://habr.com/ru/post/126848/
4. Using SRGANs to Generate Photo-realistic Images, 2018. URL: https://hub.packtpub.com/using-srgans-to-generate-photo-realistic-images-tutorial/ (дата звернення: 10.06.2020)
5. Generative Adversarial Networks: Generate Images Using Keras, 2018. URL: https://hub.packtpub.com/generative-adversarial-networks-using-keras/ (дата звернення: 10.06.2020)
6. Neural Networks and Deep Learning, URL: http://neuralnetworksanddeeplearning.com/ (дата звернення: 10.06.2020)
7. Stéphane Mallat. Understanding deep convolutional networks, 2016. URL: https://arxiv.org/pdf/1601.04920.pdf (дата звернення: 10.06.2019 р.)
8. Joan Bruna and Stéphane Mallat. Invariant scattering convolution networks. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 35(8):1872–1886, 2013.
9. Н. Старк, Р. Оску, Відновлення зображення з високою роздільною здатністю з масивів площин зображення, з використанням опуклих проекцій, J. Opt. Soc. Am. A: Опт. Зображення Sci. Vis. 6 (1989) 1715–1726.
10. С. Фарсіу, М.Д. Робінсон, М. Елад та ін., Швидке і надійне багатокадрове суперроздільна здатність, IEEE Trans. Процес зображення. 13 (2004) 1327–1344.
11. Zeyde, R., Елад, М., Проттер, М .: Збільшення масштабу з одного зображення за допомогою розріджених зображень. В: Криві і поверхні, стор. 711–730 (2012).

12. ЛіКан, Y., Бозер, B., Денкер, J.S., Едерсон, D., Говард, R.E., Хаббард, W., Жакел, L.D.: Метод зворотнього поширення похибки застосовується для розпізнавання рукописного поштового індексу. Нейронні розрахунки 541–551 (1989).
13. Крижевський А., Суцкевер І., Хінтон, Г .: Класифікація ImageNet з глибокими згортковими нейронними мережами. У: Досягнення в галузі нейронної обробки інформації. 1097–1105 (2012).
14. Фрімен, W.T., Пастор, E.C., Кармішел, O.T .: Вивчення бачення на низькому рівні. Міжнародний журнал комп'ютерного бачення 40 (1), 25–47 (2000).
15. В.Д. Колобродов, Н.И. Лихолит, В.М. Тягур, Е.В. Харитоненко.: Методы повышения пространственного разрешения тепловизорных камер с матричными приемниками излучения, стор. 56-62 (2014).
16. Пашін В. П. Методичні вказівки до виконання економіко-організаційного розділу дипломних проектів (робіт) бакалаврів і спеціалістів для студентів Інституту прикладного системного аналізу: Навч. посібник / Пашін В. П., Романов В. В., Єгорова Н. В – К.: НТУУ “КПІ”, 2011. – 118 с.

ДОДАТОК А


ДИПЛОМНА РОБОТА НА ТЕМУ
Система покращення якості зображення за допомогою генеративної змагальної мережі
Виконала: Єремєєва Вероніка, КА-65
Науковий керівник: проф. Зайченко О.Ю.


ВСТУП
ПРЕДМЕТ ДОСЛІДЖЕННЯ
Інтерполяційні методи для збільшення роздільної здатності зображення; генеративно-змагальна мережа.
ОБ'ЄКТ ДОСЛІДЖЕННЯ
Зображення низької роздільної здатності, малого розміру.
МЕТА ДОСЛІДЖЕННЯ
Дослідити існуючі методи покращення якості зображень, створити власну модель SRGAN

АКТУАЛЬНІСТЬ РОБОТИ

ВІДНОЛЕННЯ ЯКОСТІ ЗОБРАЖЕННЯ НА СТАРИХ ФОТО-ВІДЕО МАТЕРІАЛАХ

ПРИСТОСУВАННЯ ЗОБРАЖЕННЯ ДО РІЗНИХ РОЗМІРІВ ЕКРАНІВ

ПОКРАЩЕННЯ ЯКОСТІ ЗОБРАЖЕННЯ У КАМЕРХ ВІДЕОНАГЛЯДУ

ПОСТАНОВКА ЗАДАЧІ

ПРОВЕСТИ АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ПОКРАЩЕННЯ ЯКОСТІ ЗОБРАЖЕНЬ

РОЗРОБИТИ ВЛАСНУ ГЕНЕРАТИВНО-ЗМАГАЛЬНУ МЕРЕЖУ

ПРОАНАЛІЗУВАТИ ОТПРИМАНІ РЕЗУЛЬТАТИ

Основні методи обробки зображень

МЕТОД НАЙБЛИЖЧОГО СУСІДА

У алгоритмі просто дублюється значення пікселя, тобто враховується лише один піксель.

БІЛІНІЙНИЙ МЕТОД

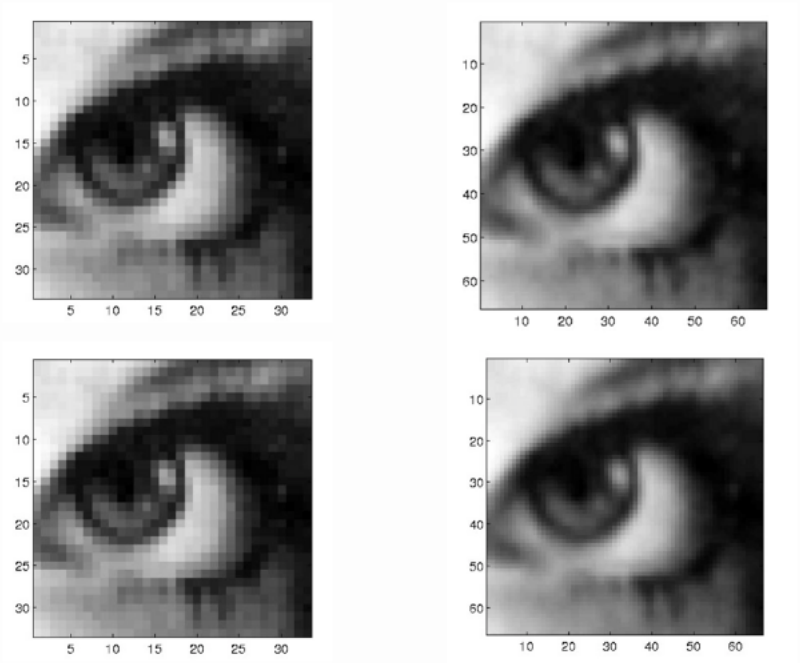
Аналізує квадрат 2x2 навколо невідомого пікселя. Значення, що інтерполюється є взважене усереднене значення.

БІКУБІЧНИЙ МЕТОД

Інтерполює масив 4x4 навколо невідомого. Найбличі пікселі мають більшу вагу, дальні - меншу.

ФІЛЬТР ЛАНЦОША

Використовує 36 пікселів для інтерполяції значення.



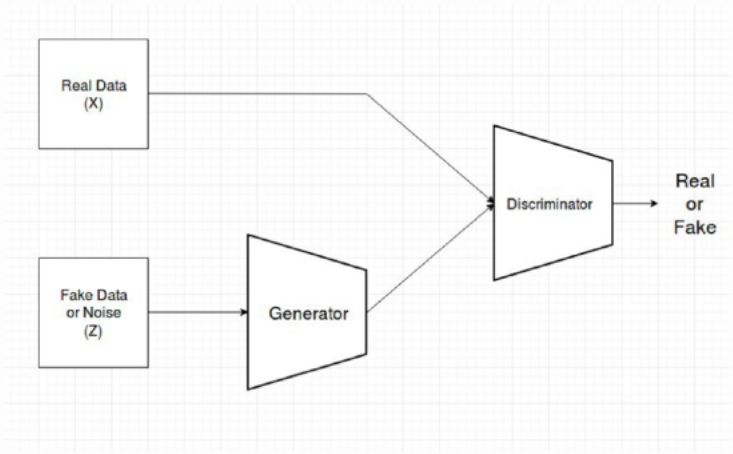
CNN

Згорткові нейронні мережі. Використовуюсть у кожному шарі мережі операцію згортки. Найчастіше використовується для задач розпізнавання/класифікації об'єктів.

GAN

Генеративна замагальна мережа. Складається з генератора та дискримінатора, які між собою утворюють мінімаксну гру.

СТРУКТУРА GAN

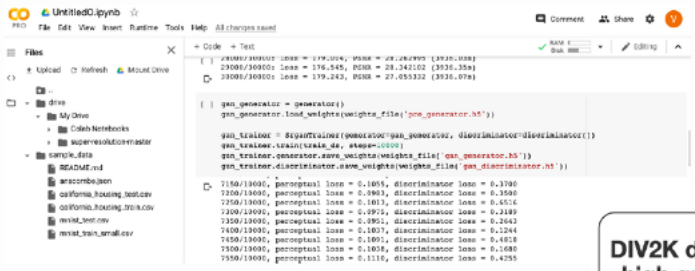


ФУНКЦІЯ ВТРАТ

$$l^{SR} = \underbrace{\underbrace{l_{\mathrm{X}}^{SR}}_{\text{content loss}} + \underbrace{10^{-3} l_{Gen}^{SR}}_{\text{adversarial loss}}}_{\text{perceptual loss (for VGG based content losses)}}$$

$$l_{Gen}^{SR} = \sum_{n=1}^{N} -\log D_{\theta_D}(G_{\theta_G}(I^{LR}))$$

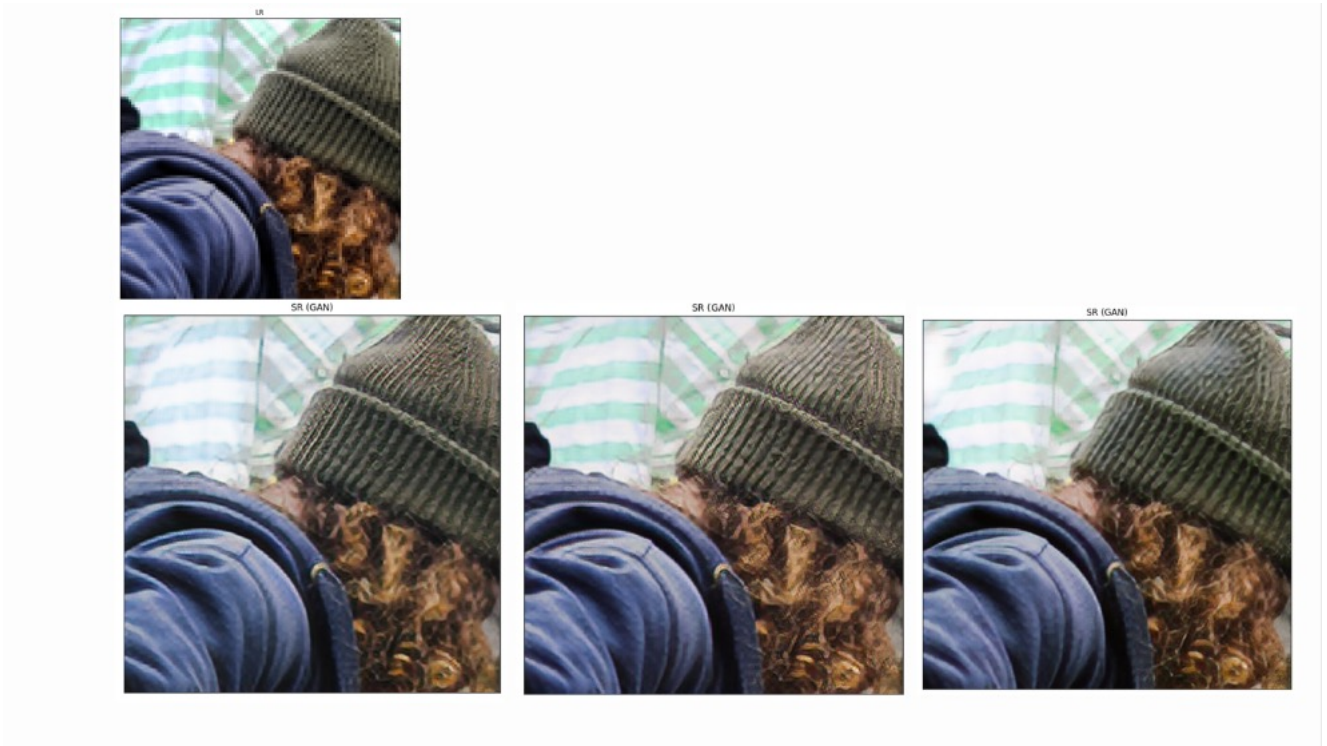
ОБРАНІ ТЕХНОЛОГІЇ



DIV2K dataset: DIVerse 2K resolution high quality images as used for the challenges @ NTIRE (CVPR 2017 and CVPR 2018) and @ PIRM (ECCV 2018)

Radu Timofte, Eirikur Agustsson, Shuhang Gu, Jiqing Wu, Andrey Ignatov, Luc Van Gool


LR
РЕЗУЛЬТАТИ РОБОТИ
SR (GAN)
SR (GAN)
SR (GAN)


LR
SR (GAN)
SR (GAN)
SR (GAN)

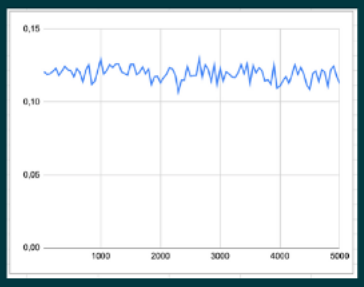
АНАЛІЗ РЕЗУЛЬТАТІВ

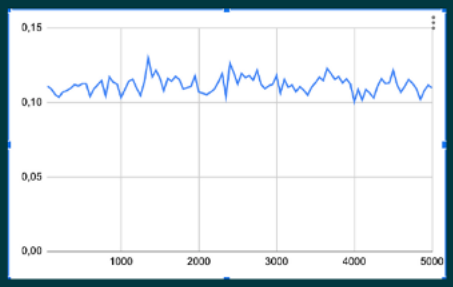
Зображення після першого тренування ставало не таким насиченим, більш сірим

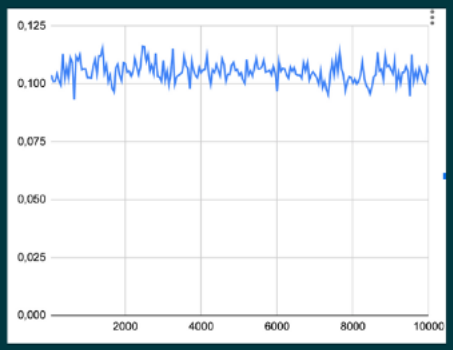
З кожним новим тренуванням мережі, якість зображення підвищується. Кольори стають більш насиченими та близькими до оригіналу

На останній ітерації тренування, дуже гарно помітно як система розпізнає текстури та краї. Вони стають нерозмитими, а навпаки більш чіткішими.

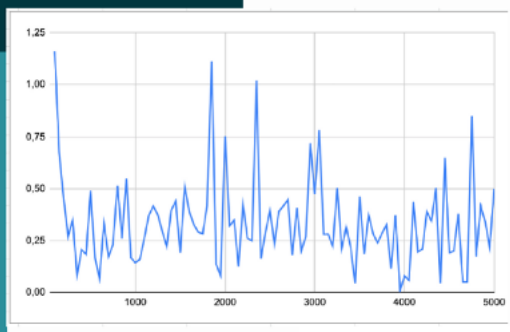
PERSEPTUAL LOSS

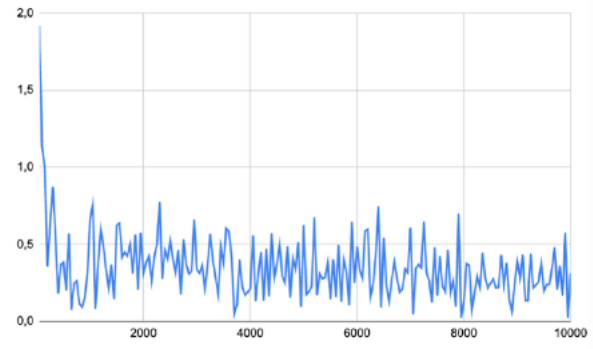






DESCRIMINATOR LOSS





РЕЗУЛЬТАТИ

З підвищенням кількості кроків тренування мережі, покращується текстурність зображення

Зображення за допомогою ГАН стають більш чіткими та мінімально розмитими

З кожним тренуванням системи функція втрат стає мешншою, що означає що помилки генератора та дискримінатора мінімізуються

Подальші дослідження

ЗБІЛЬШЕННЯ КІЛЬКОСТІ ЕПОХ ДЛЯ ТРЕНУВАННЯ КОЖНОЇ МЕРЕЖІ, ЩО ЗАБЕЗПЕЧИТЬ КРАЩІ РЕЗУЛЬТАТИ

РОЗРОБКА ПРОГРАМНОГО ДОДАТКУ ДЛЯ МОБІЛЬНОГО ДЕВАЙСУ, АБО ПК

ВИКОРИСТАННЯ ГЕНЕРАТИВНО ЗМАГАЛЬНОЇ МЕРЕЖІ ДЛЯ ВІДЕО МАТЕРІАЛІВ

ВИСНОВКИ

ПРОВЕДЕНО АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ІНТЕРПОЛЮВАННЯ ЗОБРАЖЕНЬ

РОЗРОБЛЕНА ГЕНЕРАТИВНА ЗМАГАЛЬНА МЕРЕЖА

РЕЗУЛЬТАТИ РОБОТИ БУЛИ ПРЕДСТАВЛЕНІ ТА ПРОАНАЛІЗОВАНІ НА ОСНОВІ ФУНКЦІЇ ВТРАТ ТА ЇХНІХ ГРАФІКІВ

Дякую за увагу!

ДОДАТОК Б

data.py

```
import os
import tensorflow as tf

from tensorflow.python.data.experimental import AUTOTUNE


class DIV2K:
    def __init__(self,
                 scale=2,
                 subset='train',
                 downgrade='bicubic',
                 images_dir='.div2k/images',
                 caches_dir='.div2k/caches'):

        self._ntire_2018 = True

        _scales = [2, 3, 4, 8]

        if scale in _scales:
            self.scale = scale
        else:
            raise ValueError(f'scale must be in ${_scales}')

        if subset == 'train':
            self.image_ids = range(1, 801)
        elif subset == 'valid':
            self.image_ids = range(801, 901)
        else:
            raise ValueError("subset must be 'train' or 'valid'")

        _downgrades_a = ['bicubic', 'unknown']
        _downgrades_b = ['mild', 'difficult']

        if scale == 8 and downgrade != 'bicubic':
            raise ValueError(f'scale 8 only allowed for bicubic downgrade')

        if downgrade in _downgrades_b and scale != 4:
            raise ValueError(f'{downgrade} downgrade requires scale 4')

        if downgrade == 'bicubic' and scale == 8:
            self.downgrade = 'x8'
        elif downgrade in _downgrades_b:
            self.downgrade = downgrade
        else:
            self.downgrade = downgrade
            self._ntire_2018 = False

        self.subset = subset
        self.images_dir = images_dir
```

```
        self.caches_dir = caches_dir

        os.makedirs(images_dir, exist_ok=True)
        os.makedirs(caches_dir, exist_ok=True)

    def __len__(self):
        return len(self.image_ids)

    def dataset(self, batch_size=16, repeat_count=None, random_transform=True):
        ds = tf.data.Dataset.zip((self.lr_dataset(), self.hr_dataset()))
        if random_transform:
            ds = ds.map(lambda lr, hr: random_crop(lr, hr, scale=self.scale),
num_parallel_calls=AUTOTUNE)
            ds = ds.map(random_rotate, num_parallel_calls=AUTOTUNE)
            ds = ds.map(random_flip, num_parallel_calls=AUTOTUNE)
        ds = ds.batch(batch_size)
        ds = ds.repeat(repeat_count)
        ds = ds.prefetch(buffer_size=AUTOTUNE)
        return ds

    def hr_dataset(self):
        if not os.path.exists(self._hr_images_dir()):
            download_archive(self._hr_images_archive(), self.images_dir, extract=True)

        ds = self._images_dataset(self._hr_image_files()).cache(self._hr_cache_file())

        if not os.path.exists(self._hr_cache_index()):
            self._populate_cache(ds, self._hr_cache_file())

        return ds

    def lr_dataset(self):
        if not os.path.exists(self._lr_images_dir()):
            download_archive(self._lr_images_archive(), self.images_dir, extract=True)

        ds = self._images_dataset(self._lr_image_files()).cache(self._lr_cache_file())

        if not os.path.exists(self._lr_cache_index()):
            self._populate_cache(ds, self._lr_cache_file())

        return ds

    def _hr_cache_file(self):
        return os.path.join(self.caches_dir, f'DIV2K_{self.subset}_HR.cache')

    def _lr_cache_file(self):
        return os.path.join(self.caches_dir,
f'DIV2K_{self.subset}_LR_{self.downgrade}_X{self.scale}.cache')

    def _hr_cache_index(self):
        return f'{self._hr_cache_file()}.index'
```

```
    def _lr_cache_index(self):
        return f'{self._lr_cache_file()}.index'

    def _hr_image_files(self):
        images_dir = self._hr_images_dir()
        return [os.path.join(images_dir, f'{image_id:04}.png') for image_id in self.image_ids]

    def _lr_image_files(self):
        images_dir = self._lr_images_dir()
        return [os.path.join(images_dir, self._lr_image_file(image_id)) for image_id in self.image_ids]

    def _lr_image_file(self, image_id):
        if not self._ntire_2018 or self.scale == 8:
            return f'{image_id:04}x{self.scale}.png'
        else:
            return f'{image_id:04}x{self.scale}{self.downgrade[0]}.png'

    def _hr_images_dir(self):
        return os.path.join(self.images_dir, f'DIV2K_{self.subset}_HR')

    def _lr_images_dir(self):
        if self._ntire_2018:
            return os.path.join(self.images_dir, f'DIV2K_{self.subset}_LR_{self.downgrade}')
        else:
            return os.path.join(self.images_dir, f'DIV2K_{self.subset}_LR_{self.downgrade}',
f'X{self.scale}')

    def _hr_images_archive(self):
        return f'DIV2K_{self.subset}_HR.zip'

    def _lr_images_archive(self):
        if self._ntire_2018:
            return f'DIV2K_{self.subset}_LR_{self.downgrade}.zip'
        else:
            return f'DIV2K_{self.subset}_LR_{self.downgrade}_X{self.scale}.zip'

    @staticmethod
    def _images_dataset(image_files):
        ds = tf.data.Dataset.from_tensor_slices(image_files)
        ds = ds.map(tf.io.read_file)
        ds = ds.map(lambda x: tf.image.decode_png(x, channels=3), num_parallel_calls=AUTOTUNE)
        return ds

    @staticmethod
    def _populate_cache(ds, cache_file):
        print(f'Caching decoded images in {cache_file} ...')
        for _ in ds: pass
        print(f'Cached decoded images in {cache_file}.')


# ------------------------------------------------------------
#  Transformations
```

```
# ----------------------------------------------------------


def random_crop(lr_img, hr_img, hr_crop_size=96, scale=2):
    lr_crop_size = hr_crop_size // scale
    lr_img_shape = tf.shape(lr_img)[:2]

    lr_w = tf.random.uniform(shape=(), maxval=lr_img_shape[1] - lr_crop_size + 1, dtype=tf.int32)
    lr_h = tf.random.uniform(shape=(), maxval=lr_img_shape[0] - lr_crop_size + 1, dtype=tf.int32)

    hr_w = lr_w * scale
    hr_h = lr_h * scale

    lr_img_cropped = lr_img[lr_h:lr_h + lr_crop_size, lr_w:lr_w + lr_crop_size]
    hr_img_cropped = hr_img[hr_h:hr_h + hr_crop_size, hr_w:hr_w + hr_crop_size]

    return lr_img_cropped, hr_img_cropped


def random_flip(lr_img, hr_img):
    rn = tf.random.uniform(shape=(), maxval=1)
    return tf.cond(rn < 0.5,
                   lambda: (lr_img, hr_img),
                   lambda: (tf.image.flip_left_right(lr_img),
                            tf.image.flip_left_right(hr_img)))


def random_rotate(lr_img, hr_img):
    rn = tf.random.uniform(shape=(), maxval=4, dtype=tf.int32)
    return tf.image.rot90(lr_img, rn), tf.image.rot90(hr_img, rn)



def download_archive(file, target_dir, extract=True):
    source_url = f'http://data.vision.ee.ethz.ch/cvl/DIV2K/{file}'
    target_dir = os.path.abspath(target_dir)
    tf.keras.utils.get_file(file, source_url, cache_subdir=target_dir, extract=extract)
    os.remove(os.path.join(target_dir, file))
```

com.py

```
import numpy as np
import tensorflow as tf

DIV2K_RGB_MEAN = np.array([0.4488, 0.4371, 0.4040]) * 255

def resolve_single(model, lr):
    return resolve(model, tf.expand_dims(lr, axis=0))[0]


def resolve(model, lr_batch):
```

```
    lr_batch = tf.cast(lr_batch, tf.float32)
    sr_batch = model(lr_batch)
    sr_batch = tf.clip_by_value(sr_batch, 0, 255)
    sr_batch = tf.round(sr_batch)
    sr_batch = tf.cast(sr_batch, tf.uint8)
    return sr_batch


def evaluate(model, dataset):
    psnr_values = []
    for lr, hr in dataset:
        sr = resolve(model, lr)
        psnr_value = psnr(hr, sr)[0]
        psnr_values.append(psnr_value)
    return tf.reduce_mean(psnr_values)


# ----------------------------------------
#  Normalization
# ----------------------------------------


def normalize(x, rgb_mean=DIV2K_RGB_MEAN):
    return (x - rgb_mean) / 127.5


def denormalize(x, rgb_mean=DIV2K_RGB_MEAN):
    return x * 127.5 + rgb_mean


def normalize_01(x):
    """Normalizes RGB images to [0, 1]."""
    return x / 255.0


def normalize_m11(x):
    """Normalizes RGB images to [-1, 1]."""
    return x / 127.5 - 1


def denormalize_m11(x):
    """Inverse of normalize_m11."""
    return (x + 1) * 127.5


# ----------------------------------------
#  Metrics
# ----------------------------------------


def psnr(x1, x2):
    return tf.image.psnr(x1, x2, max_val=255)
```

```
def pixel_shuffle(scale):
   return lambda x: tf.nn.depth_to_space(x, scale)
```

utils.py

```
import numpy as np
import matplotlib.pyplot as plt

from PIL import Image


def load_image(path):
   return np.array(Image.open(path))


def plot_sample(lr, sr):
   plt.figure(figsize=(20, 10))

   images = [lr, sr]
   titles = ['LR', f'SR (x{sr.shape[0] // lr.shape[0]})']

   for i, (img, title) in enumerate(zip(images, titles)):
      plt.subplot(1, 2, i+1)
      plt.imshow(img)
      plt.title(title)
      plt.xticks([])
      plt.yticks([])
```

train.py

```
import time
import tensorflow as tf

from model import evaluate
from model import srgan

from tensorflow.keras.applications.vgg19 import preprocess_input
from tensorflow.keras.losses import BinaryCrossentropy
from tensorflow.keras.losses import MeanAbsoluteError
from tensorflow.keras.losses import MeanSquaredError
from tensorflow.keras.metrics import Mean
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.optimizers.schedules import PiecewiseConstantDecay


class Trainer:
   def __init__(self,
                model,
```

```
            loss,
            learning_rate,
            checkpoint_dir='./ckpt/edsr'):

        self.now = None
        self.loss = loss
        self.checkpoint = tf.train.Checkpoint(step=tf.Variable(0),
                                              psnr=tf.Variable(-1.0),
                                              optimizer=Adam(learning_rate),
                                              model=model)
        self.checkpoint_manager = tf.train.CheckpointManager(checkpoint=self.checkpoint,
                                                             directory=checkpoint_dir,
                                                             max_to_keep=3)

        self.restore()

    @property
    def model(self):
        return self.checkpoint.model

    def train(self, train_dataset, valid_dataset, steps, evaluate_every=1000, save_best_only=False):
        loss_mean = Mean()

        ckpt_mgr = self.checkpoint_manager
        ckpt = self.checkpoint

        self.now = time.perf_counter()

        for lr, hr in train_dataset.take(steps - ckpt.step.numpy()):
            ckpt.step.assign_add(1)
            step = ckpt.step.numpy()

            loss = self.train_step(lr, hr)
            loss_mean(loss)

            if step % evaluate_every == 0:
                loss_value = loss_mean.result()
                loss_mean.reset_states()

                # Compute PSNR on validation dataset
                psnr_value = self.evaluate(valid_dataset)

                duration = time.perf_counter() - self.now
                print(f'{step}/{steps}: loss = {loss_value.numpy():.3f}, PSNR = {psnr_value.numpy():3f}
({duration:.2f}s)')

                if save_best_only and psnr_value <= ckpt.psnr:
                    self.now = time.perf_counter()
                    # skip saving checkpoint, no PSNR improvement
                    continue

                ckpt.psnr = psnr_value
```

```
                ckpt_mgr.save()

                self.now = time.perf_counter()

    @tf.function
    def train_step(self, lr, hr):
        with tf.GradientTape() as tape:
            lr = tf.cast(lr, tf.float32)
            hr = tf.cast(hr, tf.float32)

            sr = self.checkpoint.model(lr, training=True)
            loss_value = self.loss(hr, sr)

        gradients = tape.gradient(loss_value, self.checkpoint.model.trainable_variables)
        self.checkpoint.optimizer.apply_gradients(zip(gradients,
self.checkpoint.model.trainable_variables))

        return loss_value

    def evaluate(self, dataset):
        return evaluate(self.checkpoint.model, dataset)

    def restore(self):
        if self.checkpoint_manager.latest_checkpoint:
            self.checkpoint.restore(self.checkpoint_manager.latest_checkpoint)
            print(f'Model restored from checkpoint at step {self.checkpoint.step.numpy()}.')



class SrganGeneratorTrainer(Trainer):
    def __init__(self,
                 model,
                 checkpoint_dir,
                 learning_rate=1e-4):
        super().__init__(model, loss=MeanSquaredError(), learning_rate=learning_rate,
checkpoint_dir=checkpoint_dir)

    def train(self, train_dataset, valid_dataset, steps=1000000, evaluate_every=1000,
save_best_only=True):
        super().train(train_dataset, valid_dataset, steps, evaluate_every, save_best_only)


class SrganTrainer:
    #
    # TODO: model and optimizer checkpoints
    #
    def __init__(self,
                 generator,
                 discriminator,
                 content_loss='VGG54',
                 learning_rate=PiecewiseConstantDecay(boundaries=[100000], values=[1e-4, 1e-5])):
```

```
        if content_loss == 'VGG22':
            self.vgg = srgan.vgg_22()
        elif content_loss == 'VGG54':
            self.vgg = srgan.vgg_54()
        else:
            raise ValueError("content_loss must be either 'VGG22' or 'VGG54'")

        self.content_loss = content_loss
        self.generator = generator
        self.discriminator = discriminator
        self.generator_optimizer = Adam(learning_rate=learning_rate)
        self.discriminator_optimizer = Adam(learning_rate=learning_rate)

        self.binary_cross_entropy = BinaryCrossentropy(from_logits=False)
        self.mean_squared_error = MeanSquaredError()

    def train(self, train_dataset, steps=200000):
        pls_metric = Mean()
        dls_metric = Mean()
        step = 0

        for lr, hr in train_dataset.take(steps):
            step += 1

            pl, dl = self.train_step(lr, hr)
            pls_metric(pl)
            dls_metric(dl)

            if step % 50 == 0:
                print(f'{step}/{steps}, perceptual loss = {pls_metric.result():.4f}, discriminator loss =
{dls_metric.result():.4f}')
                pls_metric.reset_states()
                dls_metric.reset_states()

    @tf.function
    def train_step(self, lr, hr):
        with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
            lr = tf.cast(lr, tf.float32)
            hr = tf.cast(hr, tf.float32)

            sr = self.generator(lr, training=True)

            hr_output = self.discriminator(hr, training=True)
            sr_output = self.discriminator(sr, training=True)

            con_loss = self._content_loss(hr, sr)
            gen_loss = self._generator_loss(sr_output)
            perc_loss = con_loss + 0.001 * gen_loss
            disc_loss = self._discriminator_loss(hr_output, sr_output)

        gradients_of_generator = gen_tape.gradient(perc_loss, self.generator.trainable_variables)
        gradients_of_discriminator = disc_tape.gradient(disc_loss, self.discriminator.trainable_variables)
```

```
        self.generator_optimizer.apply_gradients(zip(gradients_of_generator,
self.generator.trainable_variables))
        self.discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator,
self.discriminator.trainable_variables))

        return perc_loss, disc_loss

    @tf.function
    def _content_loss(self, hr, sr):
        sr = preprocess_input(sr)
        hr = preprocess_input(hr)
        sr_features = self.vgg(sr) / 12.75
        hr_features = self.vgg(hr) / 12.75
        return self.mean_squared_error(hr_features, sr_features)

    def _generator_loss(self, sr_out):
        return self.binary_cross_entropy(tf.ones_like(sr_out), sr_out)

    def _discriminator_loss(self, hr_out, sr_out):
        hr_loss = self.binary_cross_entropy(tf.ones_like(hr_out), hr_out)
        sr_loss = self.binary_cross_entropy(tf.zeros_like(sr_out), sr_out)
        return hr_loss + sr_loss
```

model.py

```
from tensorflow.python.keras.layers import Add, BatchNormalization, Conv2D, Dense, Flatten, Input,
LeakyReLU, PReLU, Lambda
from tensorflow.python.keras.models import Model
from tensorflow.python.keras.applications.vgg19 import VGG19

from model.common import pixel_shuffle, normalize_01, normalize_m11, denormalize_m11

LR_SIZE = 24
HR_SIZE = 96


def upsample(x_in, num_filters):
    x = Conv2D(num_filters, kernel_size=3, padding='same')(x_in)
    x = Lambda(pixel_shuffle(scale=2))(x)
    return PReLU(shared_axes=[1, 2])(x)


def res_block(x_in, num_filters, momentum=0.8):
    x = Conv2D(num_filters, kernel_size=3, padding='same')(x_in)
    x = BatchNormalization(momentum=momentum)(x)
    x = PReLU(shared_axes=[1, 2])(x)
    x = Conv2D(num_filters, kernel_size=3, padding='same')(x)
    x = BatchNormalization(momentum=momentum)(x)
    x = Add()([x_in, x])
    return x
```

```
def sr_resnet(num_filters=64, num_res_blocks=16):
    x_in = Input(shape=(None, None, 3))
    x = Lambda(normalize_01)(x_in)

    x = Conv2D(num_filters, kernel_size=9, padding='same')(x)
    x = x_1 = PReLU(shared_axes=[1, 2])(x)

    for _ in range(num_res_blocks):
        x = res_block(x, num_filters)

    x = Conv2D(num_filters, kernel_size=3, padding='same')(x)
    x = BatchNormalization()(x)
    x = Add()([x_1, x])

    x = upsample(x, num_filters * 4)
    x = upsample(x, num_filters * 4)

    x = Conv2D(3, kernel_size=9, padding='same', activation='tanh')(x)
    x = Lambda(denormalize_m11)(x)

    return Model(x_in, x)


generator = sr_resnet


def discriminator_block(x_in, num_filters, strides=1, batchnorm=True, momentum=0.8):
    x = Conv2D(num_filters, kernel_size=3, strides=strides, padding='same')(x_in)
    if batchnorm:
        x = BatchNormalization(momentum=momentum)(x)
    return LeakyReLU(alpha=0.2)(x)


def discriminator(num_filters=64):
    x_in = Input(shape=(HR_SIZE, HR_SIZE, 3))
    x = Lambda(normalize_m11)(x_in)

    x = discriminator_block(x, num_filters, batchnorm=False)
    x = discriminator_block(x, num_filters, strides=2)

    x = discriminator_block(x, num_filters * 2)
    x = discriminator_block(x, num_filters * 2, strides=2)

    x = discriminator_block(x, num_filters * 4)
    x = discriminator_block(x, num_filters * 4, strides=2)

    x = discriminator_block(x, num_filters * 8)
    x = discriminator_block(x, num_filters * 8, strides=2)

    x = Flatten()(x)
```

```
    x = Dense(1024)(x)
    x = LeakyReLU(alpha=0.2)(x)
    x = Dense(1, activation='sigmoid')(x)

    return Model(x_in, x)


def vgg_22():
    return _vgg(5)


def vgg_54():
    return _vgg(20)


def _vgg(output_layer):
    vgg = VGG19(input_shape=(None, None, 3), include_top=False)
    return Model(vgg.input, vgg.layers[output_layer].output)
```

Notebook.ipynb

```
#Google drive connection

from google.colab import drive
drive.mount('/content/drive/')

import os
import matplotlib.pyplot as plt

from data import DIV2K
from model.srgan import generator, discriminator
from train import SrganTrainer, SrganGeneratorTrainer

%matplotlib inline
import os
weights_dir = 'weights/srgan'
weights_file = lambda filename: os.path.join(weights_dir, filename)

os.makedirs(weights_dir, exist_ok=True)

#downloading dataset

div2k_train = DIV2K(scale=4, subset='train', downgrade='bicubic')
div2k_valid = DIV2K(scale=4, subset='valid', downgrade='bicubic')
train_ds = div2k_train.dataset(batch_size=16, random_transform=True)
valid_ds = div2k_valid.dataset(batch_size=16, random_transform=True, repeat_count=1)

#pre-training of generator
pre_trainer = SrganGeneratorTrainer(model=generator(), checkpoint_dir=f'.ckpt/pre_generator')
pre_trainer.train(train_ds,
```

```
            valid_ds.take(10),
            steps=30000,
            evaluate_every=1000,
            save_best_only=False)

pre_trainer.model.save_weights(weights_file('pre_generator.h5'))


#training generator and discriminator
gan_generator = generator()
gan_generator.load_weights(weights_file('pre_generator.h5'))

gan_trainer = SrganTrainer(generator=gan_generator, discriminator=discriminator())
gan_trainer.train(train_ds, steps=10000)
gan_trainer.generator.save_weights(weights_file('gan_generator.h5'))
gan_trainer.discriminator.save_weights(weights_file('gan_discriminator.h5'))

#loading weights
pre_generator = generator()
gan_generator = generator()

pre_generator.load_weights(weights_file('pre_generator.h5'))
gan_generator.load_weights(weights_file('gan_generator.h5'))

from model import resolve_single
from utils import load_image
import PIL
from PIL import Image

def resolve_and_plot(lr_image_path):
    lr = load_image(lr_image_path)

    pre_sr = resolve_single(pre_generator, lr)
    gan_sr = resolve_single(gan_generator, lr)

    plt.figure(figsize=(20, 20))

    images = [lr, pre_sr, gan_sr]
    titles = ['LR', 'SR (PRE)', 'SR (GAN)']
    positions = [1, 3, 4]


    for i, (img, title, pos) in enumerate(zip(images, titles, positions)):
        plt.subplot(2, 2, pos)
        plt.imshow(img)
        plt.title(title)
        plt.xticks([])
        plt.yticks([])


resolve_and_plot("cropped/8c.png")
```

```
#cropping the image
import PIL
from PIL import Image

im1 = Image.open('8t.png')

left = 50
top = 20
right = 150
bottom = 120

im1 = im1.crop((left, top, right, bottom))

im1.save("8c.png")

#saving image in rgb from rgba
import PIL
from PIL import Image

rgba_image = Image.open('demo/0925-croped.png')
rgb_image = rgba_image.convert('RGB')
rgb_image.save("demo/0925q-new.png")

resolve_and_plot('demo/0925q-new.png')
```